Chapter XXIII

Computer Vision Update

Robert M. Haralick—University of Washington
Alan K. Mackworth—University of British Columbia
Steven L. Tanimoto—University of Washington

CHAPTER XXIII: COMPUTER VISION UPDATE

A. OVERVIEW

COMPUTER VISION is the science and technology of obtaining models, meanings, and control information from visual data. Inputs of a computer vision system typically are scanner outputs (usually in the form of digital images), range finder outputs, or images reconstructed by medical imaging equipment. Vision science and technology have grown more and more varied in recent years. The range of applications has been widening, and it includes many uses in manufacturing, medicine, and remote sensing.

As with artificial intelligence in general, work in vision falls mainly into two camps. The first kind of work seeks a coherent theory of visual perception and understanding (this approach is called *computational vision*), and researchers in this group often develop computational models of biological vision processes. The second camp does research and development directed toward useful applications (this is sometimes called *machine vision*). Their emphasis is on working, economical solutions to industrial, medical, and military problems rather than on the discovery of new theories or knowledge about human perception.

Perhaps the most significant development of the last five years in computational vision has been the emergence of *regularization theory* as a means for making mathematically ill-posed surface-inference problems well posed. This technique has applications in many kinds of vision problems, including reconstructing intensity maps from a limited set of samples, analyzing stereo pairs of images, and computing optical flow in dynamic imagery.

During the same period, the applications side of the field has seen important advances in three-dimensional modeling and model construction, experience with methods like those of "mathematical morphology," (resulting in better methodologies for applying such techniques), and exciting improvements in parallel computer architectures tailored to vision applications.

There has also been an interplay between computational vision and machine vision. The stereo algorithms, developed largely within the computational vision camp, have moved out into the realm of industrial application. Computer architectures, developed with industrial vision in mind, are influencing studies in computational vision, for example, in the development of parallel algorithms for solving reconstruction problems on meshes.

Because of the large amount of activity in these two areas of com-

puter vision, and because of page limitations here, the scope of this chapter is necessarily limited. The major advances in low-level vision, computational vision, and vision architectures are emphasized. Relatively little is said about specific software implementations (this is in contrast to the vision coverage in Chapter XIII of the *Handbook*).

B. LOW-LEVEL VISION

UNDERLYING some approaches to computational vision and to machine vision are basic tasks of breaking up an image into component regions. This *segmentation* problem must be tackled before determining 3-D surface characteristics or recognizing objects in the scene. A large variety of methods have been invented and studied for this initial analysis task. The next subsection gives an overview of this subfield, expanding upon the description of *region analysis* in Article XIII.C5 in Volume III.

B1. Segmentation Techniques

IN TRADITIONAL APPROACHES to computer vision, the pixels of an image are grouped into regions in a process called *segmentation*, and this is done prior to any attempt to interpret the regions as objects in the scene. A perceived advantage of computing a segmentation is that one could, relatively easily, achieve a relatively concise representation of the image's essential pictorial aspects, and that this would permit the semantic phase of the analysis to be accomplished painlessly. Except in certain artificial environments, segmentation has proven to be difficult in itself, and it seems that semantic considerations are often needed at the segmentation level. Nonetheless, various segmentation methods make up an important part of the arsenal of techniques that can be employed in computer vision, and they provide a good starting point for a tutorial overview of developments in vision.

What should a good image segmentation be? Although this depends largely on the application, it can be answered in an application-independent way to a certain extent. Let us attempt to do so.

Regions of an image segmentation should be homogeneous—uniform with respect to some characteristic such as gray tone or texture. Region interiors should usually be simple and without many small holes. Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they are uniform. Boundaries of each segment should be simple, not ragged, and must be spatially accurate.

Achieving all these desired properties is difficult because strictly uniform and homogeneous regions are typically full of small holes and

have ragged boundaries. Insisting that adjacent regions have large differences in values can cause regions that ought to be kept separated to merge and thus the intervening boundaries to be lost.

Just as there is no generally accepted theory of clustering in statistics, there is no well-accepted theory of image segmentation. Image segmentation techniques tend to be ad hoc. They differ in the ways in which they emphasize one or more of the desired properties and in the ways in which they balance and compromise one desired property against another.

Image segmentation techniques can be classified into one of the following groups:

1. Measurement-space–guided spatial clustering
2. Single-linkage region-growing schemes
3. Hybrid-linkage region-growing schemes
4. Centroid-linkage region-growing schemes
5. Spatial clustering schemes
6. Split-and-merge schemes

As this brief typology suggests, image segmentation can be viewed as a clustering process. The difference between image segmentation and clustering is in grouping. In clustering, the grouping is done in measurement space (e.g., the space of gray values rather than the space of pixel coordinate pairs). In image segmentation, the grouping is done on the spatial domain of the image, and there is an interplay in the clustering between the (possibly overlapping) groups in measurement space and the mutually exclusive groups of the image segmentation.

The single-linkage region-growing schemes are the simplest and most prone to the unwanted region-merge errors. The hybrid-linkage and centroid-linkage region-growing schemes are better in this regard. The split-and-merge technique is not as subject to the unwanted region-merge error. However, it suffers from large memory usage and excessively blocky region boundaries. The measurement-space–guided spatial clustering tends to avoid both the region-merge errors and the blocky boundary problems because of its primary reliance on measurement space. But the regions produced are not smoothly bounded, and they often have holes, giving the effect of salt-and-pepper noise. The spatial clustering schemes may be better in this regard, but they have not been tested well enough. The hybrid-linkage schemes appear to offer the best compromise between having smooth boundaries and few unwanted region merges.

The remainder of this section describes the main ideas behind the major image segmentation techniques. Additional image segmentation

surveys can be found in Zucker (1976), Riseman and Arbib (1977), Kanade (1980), and Fu and Mui (1981), and Haralick and Shapiro (1985).

Measurement-space–Guided Spatial Clustering

This technique for image segmentation uses the measurement-space clustering process to define a partition in measurement space (e.g., the space of pixel gray values of the image). Then each pixel is assigned the label of the cell in the measurement-space partition to which it belongs. The image segments are defined as the connected components of the pixels having the same label.

The accuracy of image segmentation using the measurement-space clustering process depends directly on how well the objects of interest on the image separate into distinct measurement-space clusters. Typically the process works well in situations where there are a few kinds of distinct objects having widely different gray-tone intensities (or gray-tone intensity vectors, for multiband images) and these objects appear on a nearly uniform background.

Clustering procedures that use the pixel as a unit and compare each pixel value with every other pixel value can require excessively large computation times because of the large number of pixels in an image. Iterative partition-rearrangement schemes such as ISODATA have to go through the image data set many times and if done without sampling can also take excessive computation time. Histogram-mode seeking, because it requires only one pass through the data, probably involves the least computation time of the measurement-space clustering techniques, and it is the one we discuss here.

Histogram-mode seeking is a measurement-space clustering process in which it is assumed that homogeneous objects on the image manifest themselves as the clusters in measurement space. Image segmentation is accomplished by mapping the clusters back to the image domain where the maximal connected components of the mapped back clusters constitute the image segments. For single-band images, calculation of this histogram in an array is direct. The measurement-space clustering can be accomplished by determining the valleys in this histogram and declaring the clusters to be the interval of values between valleys. A pixel whose value is in the ith interval is labeled with index i and the segment it belongs to is one of the connected components of all pixels whose label is i.

Ohlander et al. (1975) refines the clustering idea in a recursive way. He begins by defining a mask selecting all pixels on the image. Given any mask, a histogram of the masked image is computed. Measurement-space clustering enables the separation of one mode of the histogram set from another mode. Pixels on the image are then identified with the

cluster to which they belong. If there is only one measurement-space cluster, the mask is terminated. If multiple clusters are present, the process is repeated for each connected component (region) associated with each cluster. Note that one cluster may produce more than one connected component. During successive iterations, the next mask in the stack selects pixels in the histogram-computation process. Clustering is repeated for each new mask until the stack is empty. The process is illustrated in Figure B–1.

Single-linkage Region Growing

Single-linkage region growing schemes regard each pixel as a node in a graph. Neighboring pixels whose properties are "similar enough" are joined by an arc. The image segments are maximal sets of pixels all belonging to the same connected component. Single-linkage image-segmentation schemes are attractive for their simplicity. They do, however, have a problem with chaining, because it takes only one arc leaking from one region to a neighboring one to cause the regions to merge.

The simplest single-linkage scheme defines "similar enough" by pixel difference. Two neighboring pixels are similar enough if the absolute value of the difference between their gray-tone intensity values is small enough. Bryant (1979) defines "similar enough" by normalizing the difference by the quantity $\sqrt{2}$ times the root-mean-square value of neighboring pixel differences taken over the entire image.

For pixels having vector values, the obvious generalization is to use a vector norm of the pixel-difference vector. Instead of using a Euclidean distance, Asano and Yokoya (1981) suggest that two pixels be joined together if the absolute value of their difference is small enough compared to the average absolute value of the center pixel minus neighbor pixel for each of the neighborhoods to which the pixels belong. The ease with which unwanted region chaining can occur with this technique limits its potential on complex or noisy data.

Hybrid-linkage Region Growing

Hybrid single-linkage techniques are more powerful than the simple single-linkage technique. The hybrid techniques seek to assign a property vector to each pixel where the property vector depends on the neighborhood of the pixel. Pixels that are similar are so because their neighborhoods in some special sense are similar. Similarity is thus established as a function of neighboring pixel values, and this makes the technique better behaved on noisy data.

One hybrid single-linkage scheme relies on an edge operator to establish whether two pixels are joined with an arc. Here an edge operator is applied to the image, labeling each pixel as edge or nonedge. Neighboring

pixels, neither of which are edges, are joined by an arc. The initial segments are the connected components of the nonedge labeled pixels. The edge pixels can either be left as edges and be considered as background or they can be assigned to the spatially nearest region having a

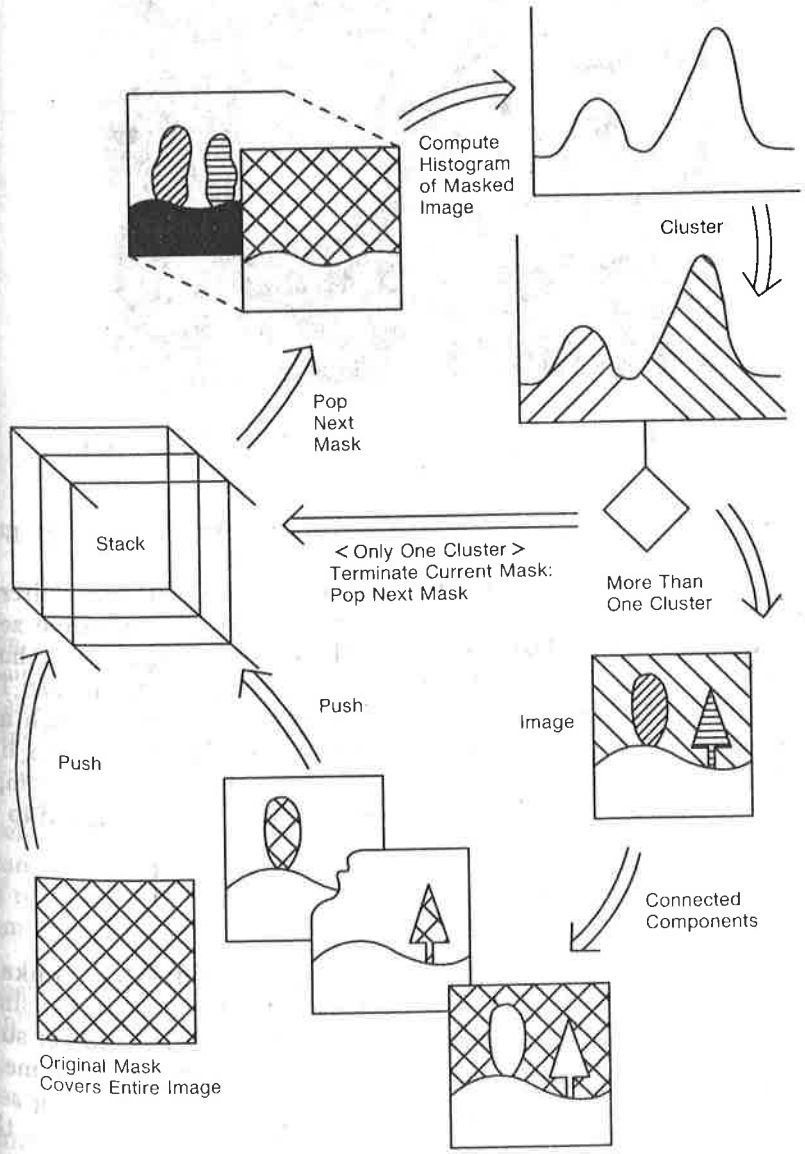

Figure B–1. The recursive histogram spatial clustering method of Ohlander.

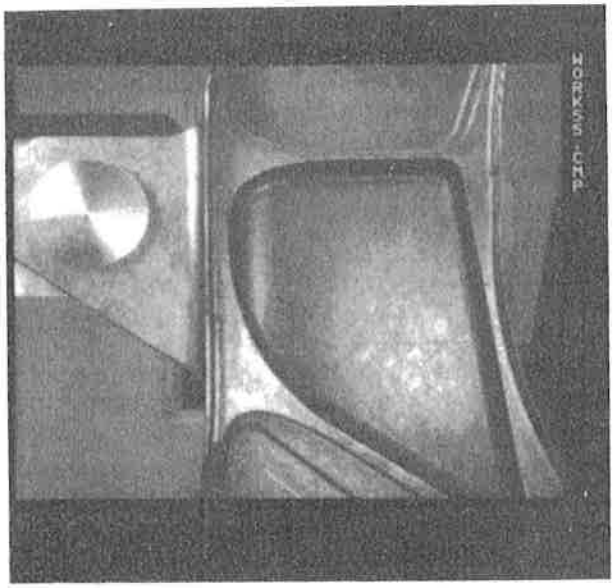

Figure B–2. Image of a bulkhead of an F-15 aircraft.

label. Successful use of this technique may require closing edge gaps before performing the region growing.

Figure B–2 illustrates an image of a section of an F-15 aircraft bulkhead. Figure B–3 illustrates a second directional derivative zero-crossing operator applied to the image of Figure B–2. Figure B–4 shows the segmentation that results from connecting the non-edge pixels. The method is thus a hybrid-linkage region-growing scheme in which any pair of neighboring pixels, neither of which are edge pixels, can link together. The resulting segmentation consists of the connected components of the nonedge pixels and where each edge pixel is assigned to its nearest connected component.

Centroid-linkage Region Growing

In centroid-linking region growing, in contrast with single-linkage region growing, pairs of neighboring pixels are not compared for similarity. Rather, the image is scanned in some predetermined manner such as left to right or top to bottom. A pixel's value is compared to the mean of an already existing but not necessarily completed neighboring segment. If its value and the segment's mean value are close enough, the pixel is added to the segment and the segment's mean is updated. If more than one region is close enough, it is added to the closest region. However, if the means of the two competing regions are close enough, the two

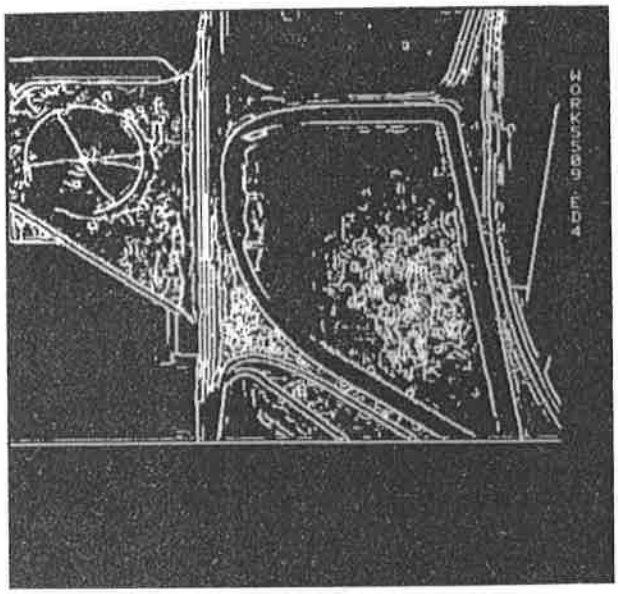

Figure B–3. Directional derivative zero-crossing operator applied to the F-15 image.

regions are merged and the pixel is added to the merged region. If no neighboring region has its mean close enough, a new segment is established having the given pixel's value as its first member. The scan geometry for the centroid-linkage region-growing scheme is shown in Figure B–5.

Keeping track of the means and scatters for all region as they are being determined does not require large amounts of memory space. There cannot be more regions active at one time than the number of pixels in a row of the image. Hence a hash table mechanism with the space of a small multiple of the number of pixels in a row can work well.

One way of performing the region growing is by the use of the T-test. Let R be a segment of N pixels neighboring a pixel with gray-tone intensity y. Define the mean $\overline{X}$ and scatter S^2 by

$$X = \frac{1}{N} \sum_{(r,c)\epsilon R} I(r, c) \tag{1}$$

and

$$S^2 = \sum_{(r,c)\epsilon R} (I(r, c) - X)^2 \tag{2}$$

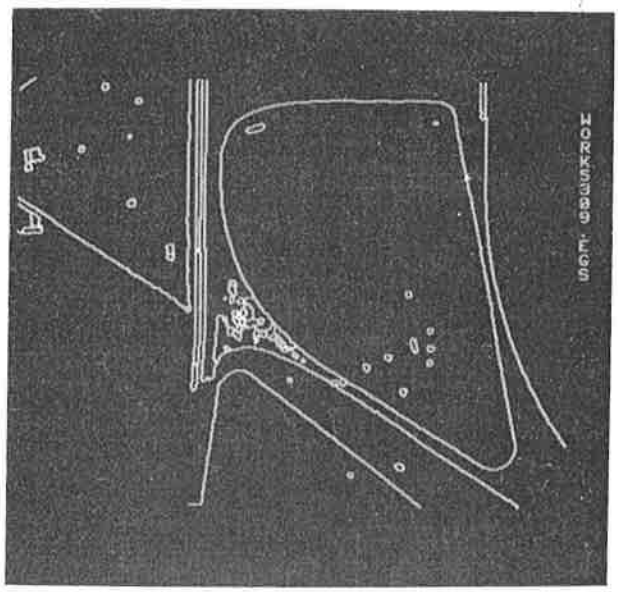

Figure B–4. Segmentation of the F-15 image.

Under the assumption that all the pixels in R and the test pixel y are independent and identically distributed normals, the statistic

$$T = \left[\frac{(N-1)N}{(N+1)}(y-\overline{X})^2/S^2\right]^{1/2} \tag{3}$$

has a T_{N-1} distribution. If T is small enough, y is added to region R and the mean and scatter are updated using y. The new mean and scatter are given by

$$\overline{X}_{\text{new}} \leftarrow (N\overline{X}_{\text{old}} + y)/(N+1) \tag{4}$$

and

$$S^2_{\text{new}} \leftarrow S^2_{\text{old}} + (y-\overline{X})^2 + N(\overline{X}_{\text{new}} - \overline{X}_{\text{old}})^2 \tag{5}$$

If T is too high, the value y is not likely to have arisen from the population of pixels in R. If y is different from all of its neighboring regions, it begins its own region. A slightly stricter linking criterion can require that not only must y be close enough to the mean of the neighboring regions, but also that a neighboring pixel in that region must have a close enough value to y. This combines a centroid linkage and single linkage criterion.

The Levine and Shaheen scheme (1981) is similar. The difference is that Levine and Shaheen attempt to keep regions more homogeneous

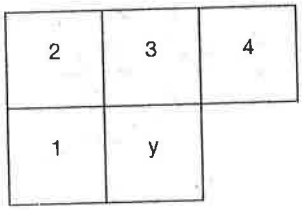

Figure B–5. Region-growing geometry for the one-pass scan, left-right, top-bottom region growing.

and try to keep the region scatter from getting too high. They do this by requiring the differences to be more significant before a merge takes place if the region scatter is high. For a user-specified value θ, they define a test statistic T where

$$T = |y - X_{\text{new}}| - (1 - S/\overline{X}_{\text{new}})\theta \tag{6}$$

If $T < 0$ for the neighboring region R in which $|y - \overline{X}|$ is the smallest, y is added to R. If $T > 0$ for the neighboring region in which $|y - \overline{X}|$ is the smallest, y begins a new region.

Figure B–6 illustrates the application of the centroid-linkage region-growing technique to the bulkhead image. This application uses two successive scans of the image. The first is a left-right top-down scan, and the second is a right-left bottom-top scan.

Hybrid-linkage Combination Techniques

The centroid-linkage and the hybrid-linkage methods can be combined in a way that takes advantage of their relative strengths. The strength of the single-linkage method is that boundaries are placed in a spatially accurate way. Its weakness is that edge gaps result in excessive merging. The strength of the centroid-linkage method is its ability to place boundaries in weak-gradient areas. It can do this because it does not depend on a large difference between the pixel and its neighbor to declare a boundary. It depends instead on a large difference between the pixel and the mean of the neighboring region to declare a boundary.

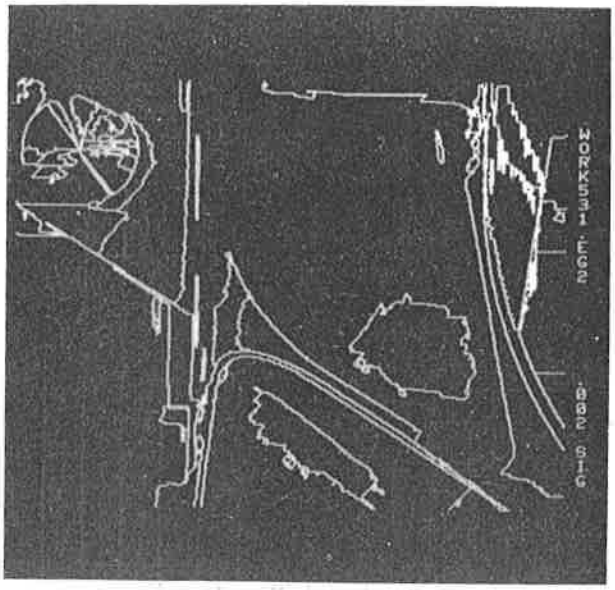

Figure B–6. The two-pass top-down centroid segmentation of the bulkhead image.

The combined centroid-hybrid linkage technique does the obvious thing. Centroid linkage is only done for nonedge pixels; that is, region growing is not permitted across edge pixels. Saying it another way, edge pixels are not permitted to be assigned to any region and cannot link to any region. Thus, if the parameters of centroid linkage were set so that any difference, however large, between pixel value and region mean was considered small enough to permit merging, the two-pass hybrid combination technique would produce a connected components of the nonedge pixels. As the difference criterion is made more strict, the centroid linkage produces boundaries in addition to those produced by the edges. Figure B–7 illustrates the application of the hybrid-linkage technique to the bulkhead image.

Split-and-Merge

The split-and-merge method for segmentation begins with the entire image as the initial segment. Then it successively splits each of its current segments into quarters if the segment is not homogeneous enough. Homogeneity can be easily established by determining if the difference between the largest and smallest gray-tone intensities is small enough. Algorithms of this type were first suggested by Robertson (1973)

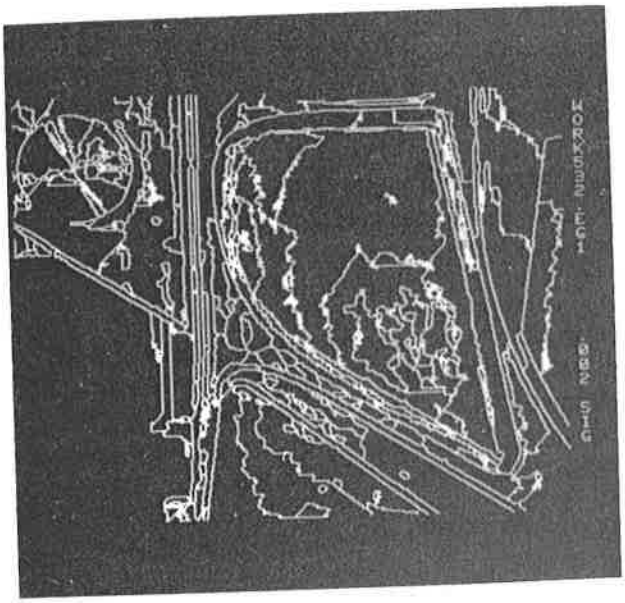

Figure B–7. Segmentation using the one-pass combined centroid and hybrid linkage method.

and Klinger (1973). Kettig and Landgrebe (1975) try to split all nonuniform 2×2 neighborhoods before beginning the region merging. Fukada (1980) suggests successively splitting a region into quarters until the sample variance is small enough. The efficiency of the split-and-merge method can be increased by arbitrarily partitioning the image into square regions of a user-selected size and then splitting these further if they are not homogeneous.

Because segments are successively divided into quarters, the boundaries produced by the split technique tend to be squarish and slightly artificial. Sometimes adjacent quarters coming from adjacent split segments need to be joined rather than remain separate. Horowitz and Pavlidis (1976) suggest a split-and-merge strategy to take care of this problem. Muerle and Allen (1968) suggest merging a pair of adjacent regions if their gray-tone intensity distributions are similar enough. They recommend the Kolmogorov-Smirnov test.

Chen and Pavlidis (1980) suggest using statistical tests for uniformity rather than a simple examination of the difference between the largest and smallest gray-tone intensities in the region under consideration for splitting. The uniformity test requires that there be no significant difference between the mean of the region and each of its quarters.

The Chen and Pavlidis tests assume that the variances are equal and known.

Let each quarter have K pixels, X_{ij} be the jth pixel in the ith region, X_i be the mean of the ith quarter, and $X..$ be the grand mean of all the pixels in the four quarters. Then, for a region to be considered homogeneous, Chen and Pavlidis require that

$$|X_i - X..| \leq \epsilon, \qquad i = 1, 2, 3, 4 \tag{7}$$

We give here the F-test for testing the hypothesis that the mean and variances of the quarters are identical. The value of variance is not assumed known. If we assume that the regions are independent and identically distributed normals, the optimal test is given by the statistic F, which is defined by

$$F = \frac{K \sum_{i=1}^{4} (X_{i\cdot} - X..)^2/3}{\sum_{i=1}^{4} \sum_{k=1}^{K} (X_{ik} - X_{i\cdot})^2/4(K-1)} \tag{8}$$

It has a $F_{3,4(K-1)}$ distribution. If F is too high, the region is declared not uniform.

The data structures required to do a split-and-merge on images larger than 512 × 512 are very large. Execution of the algorithm on virtual-memory computers results in so much paging that the dominant activity may be paging rather than segmentation. Browning and Tanimoto (1982) describe a split-and-merge scheme where the split-and-merge is first accomplished on mutually exclusive subimage blocks and the resulting segments are then merged between adjacent blocks to take care of the artificial block boundaries.

B2. Edges

IF AN IMAGE is successfully segmented into regions, the contours of the regions are available for shape analysis. However, it is sometimes more expedient to compute the contours directly from the image, rather than to go through one of the previously described segmentation processes. To compute contours directly from the image, "edge detection" must be performed. This subsection discusses the important characteristics of edges. Edge detection continues to be a subject of intense research. Elementary methods for edge detection, including the Roberts cross operator and the Sobel operator, are described in Article XIII.C4, Vol. III.

The Difficulties of Finding the Contours of Objects in an Image

What is an edge in a digital image? The first intuitive notion is that a digital edge occurs on the boundary between two pixels when the

respective brightness values of the two pixels are significantly different. "Significantly different" may depend on the distribution of brightness values around each of the pixels.

We often point to a region on an image and say this region is *brighter* than its surrounding area, meaning that the mean of the brightness values of pixels inside the region is greater than the mean of the brightness values outside the region. Having noticed this, we would then say that an *edge* exists between each pair of neighboring pixels where one pixel is inside the region and the other is outside the region. Such edges are referred to as *step edges*.

Step edges are not the only kind of edge. If we scan through a region left to right observing the brightness values steadily increasing, and then after a certain point we observe that the brightness values are steadily decreasing, we are likely to say that there is an edge at the point of change from increasing to decreasing brightness values. Such edges are called *roof edges*.

Thus, in general, an *edge* is a place in an image where there appears to be a jump in brightness value or a jump in brightness value derivative.

In some sense, this summary statement about edges is quite revealing because in a discrete array of brightness values there are jumps (in the literal sense) between neighboring brightness values if the brightness values are different, even if only slightly different. Perhaps more to the heart of the matter, there exists no definition of derivative for a discrete array of brightness values. The only way to interpret jumps in value and jumps in derivatives when referring to a discrete array of values is to assume that the discrete array of values comes about as some kind of sampling of a real-valued function defined on a bounded and connected subset of the real plane R^2. The jumps in value and jumps in derivative really must refer to points of discontinuity of f and to points of discontinuity in the partial derivatives of f.

Edge finders should then regard the digital picture function as a sampling of the underlying function f, where some kind of random noise has been added to the true function values. To do this, the edge finder must assume some kind of parametric form for the underlying function f, use the sampled brightness values of the digital picture function to estimate the parameters, and finally make decisions regarding the locations of discontinuities of the underlying function and its partial derivatives based on the estimated values of the parameters.

Of course, it is impossible to determine the true locations of discontinuities in value or derivatives based on samplings of the functions. The locations are estimated by function approximation. The location of the estimated discontinuity will be where the first derivative has a relative maximum. This is where the second derivative will have a negatively shaped zero-crossing if the edge is being crossed from low value to high value. Sharp discontinuities will reveal themselves in high values for

estimates of first partial derivatives. Sharp discontinuities in derivative will reveal themselves in high values for estimates of the second partial derivatives. This means that the best we can do is to assume that the first and second derivatives of any possible underlying image function have known bounds. Therefore any estimated first- or second-order partials that exceed these known bounds must be due to discontinuities in value of the underlying function. The location of the estimated discontinuity in derivative will be where the second derivative has a relative extremum and this will be where the third derivative has an appropriately shaped zero-crossing.

Recent Developments

Marr and Hildreth (1980) used for the second derivative the isotropic Laplacian. Haralick (1984) and Canny (1986) used, for the second derivative, the second directional derivative taken in a direction that extremizes the first directional derivative. The implementation of each of these zero-crossing edge operators is quite different.

Since the differentiation of a sampled signal is, properly speaking, an ill-posed problem, it has been proposed that edge detection be performed by first filtering the image (or "regularizing" it) and then differentiating it. A mathematical problem is *well-posed* in the sense of Hadamard, provided its solution exists, is unique, and depends continuously on the given data. Regularization refers to the transformation of an ill-posed problem into a well-posed one. Standard methods of regularization have been developed—see, for example, Tikhonov and Arsenin (1977)—and applied in edge detection. Details may be found in Torre and Poggio (1986). A good overview of edge detection, including a discussion of regularization, may be found in Hildreth (1987).

B3. Stereo

Overview

The objective in many computer vision problems is to reconstruct a three-dimensional surface representation of a scene from the image information output by cameras. Video cameras provide only 2-D images, and stereo methods must be used to obtain depth information. The use of two (or more) images of the same scene, taken from different positions, can permit the determination of depth using *parallax*—the analysis of each triangle formed by some notable surface point in the scene and the two camera viewpoints. With two such images, the method of depth determination is called *binocular stereo*. With three, it is *trinocular stereo*. With more, it is sometimes called *multiple-image stereo*. For an intro-

duction to binocular stereo, see Article XIII.D3, or see Barnard and Fischler (1987). When the scene is static but a sequence of images is taken from a moving viewpoint, *motion stereo* may be used to establish 3-D information.

The usual sequence of steps needed in binocular stereo is as follows:

1. Input images either from two cameras or from one camera at two different times and positions.
2. Determine camera parameters—position, orientation, focal length, and so on.
3. Detect/select feature points in the images that are candidates for matching (e.g., edge points).
4. Match feature points by constructing a correspondence between feature points of the two images.
5. Compute depth values at the locations of the matched feature points.
6. Interpolate depth values at all or many of the points in the image that are not locations of matched feature points.

Feature Point Detection/Selection

With a simple camera geometry we may assume that the two images of a point in the scene have a positional disparity along the x-axis of the image but not along the y-axis. To determine this disparity, using feature-based or edge-based stereo, the points must be detected in each image and then put into correspondence. Generally speaking, only certain points in the image are capable of being matched directly; these are prominent locations in the image that are easily distinguished from neighboring points. In most cases the feature points can be obtained using edge-detection methods.

A popular method for finding feature points for stereo matching requires that the Laplacian operator be applied to the image (see Volume 3, p. 211–212). Then the zero-crossing contours of the resulting image are identified. The points on the zero-crossing contours are taken as the feature points. Since the digital images have a limited number of scan lines, the number of zero-crossing points is generally manageable.

Because the disparities occur in the x direction, it is usually sufficient to perform the differentiation (or apply the Laplacian) in one dimension, along each scan line of the image. This is computationally inexpensive in comparison with two-dimensional Laplacians.

If general camera geometries are used, the feature points must be distinguishable in both the x and y directions. Although the detection of these points is therefore more computationally expensive, the resulting number of points is usually less than for one-dimensional analysis, and this can speed up the matching process. Scene points that generate good

feature points with distinction in both dimensions are corners (vertices) of polyhedra and bright spots and corners of 2-D patterns painted on the surfaces of objects in the scene.

It is also possible to match areas rather than features. In area-based matching, correspondences are typically established using cross-correlation. This tends to be computationally more expensive and also less accurate than feature-based or edge-based matching. However, area-based stereo can be more robust in cases of noisy images or images with poorly defined edges.

Matching. Although matching for stereo is similar in spirit to model matching for object recognition, it is also somewhat different. In the case of horizontally constrained displacement, we have a collection of one-dimensional matching problems, one for each scan line. We can expect the disparity function along the scan line to exhibit some coherence as we move to each successive scan line, as well as along the line. Therefore the solutions to each 1-D matching problem are not completely independent.

Some of the approaches to matching are as follows:

1. Coarse-to-fine (see Marr and Poggio, 1977; and Grimson, 1985)
2. Dynamic programming (see Baker and Binford, 1981)
3. Energy minimization (see Direct Matching with Simulated Annealing, described below)
4. Ad hoc correspondence building

Interpolating Depth Values. The problem of obtaining a full set of depth values from the sparse set obtained from feature-based stereo can be solved with interpolation. However, this interpolation should satisfy both smoothness on surfaces and maintain sudden depth changes at surface boundaries. In the case of natural terrain, quadratic surface fitting may be appropriate (see Smith, 1984). For rapid interpolation subject to smoothness constraints, multigrid methods may be used (see Section D).

Direct Matching with Simulated Annealing. A method of matching a stereo pair of images using simulated annealing has been proposed by Barnard (1987). This is an area-based rather than a feature-based approach. An energy measure E is to be minimized through the adjustment of disparity values $D_{i,j}$:

$$I = \sum_{i,j} (|\Delta I_{ij}| + \lambda |\nabla D_{ij}|)$$

where $\Delta I_{ij} = \mathrm{I}_L(i,j) - \mathrm{I}_R(i, j + D_{ij})$; I_L and I_R are the left- and right-image intensity values; and D_{ij} is the disparity value for location (i,j). This measures the difference in intensity between each two matched

points as well as the unsmoothness of the disparity function. If both of these terms are zero, the two images match perfectly, except for a translation, and the scene must be flat.

Starting from an initial high-energy state, the disparity values are adjusted stochastically according to the Metropolis algorithm (see page 576) or with an alternative method proposed by Barnard.

Nonbinocular Methods. *Trinocular stereo* employs three images of a scene to obtain 3-D surface data. The third image, taken from a viewpoint not colinear with the other two, greatly reduces the number of incorrect matches and it can increase the accuracy of the resulting depth information. A method that permits the three cameras to be in arbitrary positions is described by Ayache and Lustman (1987). One that requires the viewpoints to form a right triangle is given by Ohta et al. (1986). Others are given by Yachida et al. (1986), Ito and Ishii (1986), and Pietikainen and Harwood (1986). The number of viewpoints need not be limited to three. Multiple-image stereo allows additional improvements in accuracy at the expense of higher computational cost (see Yachida, 1985).

In addition to binocular, trinocular, and multiple-image stereo, surface orientation may be computed using two images from the same viewpoint, but taken under illumination by a light source in two different positions. This method is called *photometric stereo* and is described briefly in the Overview to Chapter XIII in Volume III of the *Handbook*. The change in shading at a surface point from one image to the other gives an indication of the surface gradient at that point. Such methods are described in Woodham (1980).

B4. Mathematical Morphology for Image Analysis

A CLASS OF TECHNIQUES called *mathematical morphology* has found a variety of applications in industrial machine vision. This section presents the primary operations of mathematical morphology: dilation, erosion, opening, and closing. In addition to their definitions, some properties of these operations are also given.

The mathematical morphology approach to the processing of digital images is based on shape. Appropriately used, these techniques can simplify image data, preserving essential shape characteristics and eliminating irrelevancies. Since the identification of objects, features, and manufacturing defects depend closely on shape, this approach is natural for such tasks.

Although the techniques are being used in the industrial world, the basis and theory of binary morphology are not covered in many texts or

monographs. Exceptions are the highly mathematical books by Matheron (1975) and Serra (1982).

The language of mathematical morphology is that of set theory. Sets in mathematical morphology represent the shapes that are manifested on binary or gray-tone images. The set of all the black pixels in a black and white image (a binary image) constitutes a complete description of the binary image. Sets in two-dimensional Euclidean space are represented by foreground regions in binary images. Sets in three-dimensional Euclidean space may actually represent time-varying binary imagery or static gray-scale imagery as well as binary solids. Sets in higher dimensional spaces may incorporate additional image information such as color, or multiple perspective imagery. Mathematical morphology transformations apply to sets of any dimensions, including those in Euclidean N-space and its discrete or digitized equivalents, the set of N-tuples of integers, Z^N. For the sake of simplicity we will refer to either of these sets as E^N.

Those points in a set being morphologically transformed are considered as the *selected set* of points, and those in the complement set are considered as not selected. Hence, morphology from this point of view is *binary* morphology. We begin our discussion with the morphological operation of dilation.

Dilation

Dilation is a morphological transformation that combines two sets using vector addition of set elements. If A and B are sets in N-space (E^N) with elements a and b, respectively, $a = (a_1, ..., a_N)$ and $b = (b_1, ..., b_N)$ being N-tuples of element coordinates, then the dilation of A by B is the set of all possible vector sums of pairs of elements, one coming from A and one coming from B. Denoting dilation by $\oplus$,

$$A \oplus B = \{c \in E^N \mid c = a + b \quad \text{for some } a \in A \text{ and } b \in B\}$$

Dilation as a set theoretic operation was proposed by Minkowski (1903) to characterize integral measures of certain open (sparse) sets. Dilation as an image-processing operation was employed by several early investigators in image processing as smoothing operations: Unger (1958), Golay (1969), and Preston (1961, 1973). Dilation as an image operator for shape extraction and estimation of image parameters was explored by Matheron (1975) and Serra (1972).

Mathematically the roles of the sets A and B are symmetric; the dilation operation is commutative because addition is commutative. Hence $A \oplus B = B \oplus A$. In practice, A and B are handled quite differently. The first operand is considered to be the image undergoing analysis, whereas the second operand, referred to as the *structuring element*, is

thought of as constituting a single shape parameter of the dilation transformation.

Dilation of a set by a structuring element in the shape of a disk results in an isotropic swelling or expansion of the set. (Approximating the disk by a small square, 3×3, the expansion can be implemented as a neighborhood operation on a mesh architecture or pipelined image-processing architecture.) Some sample dilation transformations are illustrated in Figures B–8 and B–9. In Figure B–8, the upper left is the input image consisting of a cross. The lower right shows an octagonal structuring element. The upper right shows the input image dilated by the octagonal structuring element. In Figure B–9, the upper left contains the input image consisting of two objects. The upper right shows the input image dilated by the structuring element {(0, 0), (14, 0)}. The lower left shows the input image dilated by the structuring element {(0, 0), (0, 14)}. The lower right shows the input image dilated by the structuring element {(0, 0), (14, 0), (0, 14)}. This example illustrates that dilation can be viewed as the replication of a pattern. In actual use, the replicated copies of the pattern usually overlap, as in Figure B–8.

Since addition is associative, the dilation of an image A by a structuring element D, which is itself a dilation $D = B \oplus C$, can be computed as

$$A \oplus D = A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

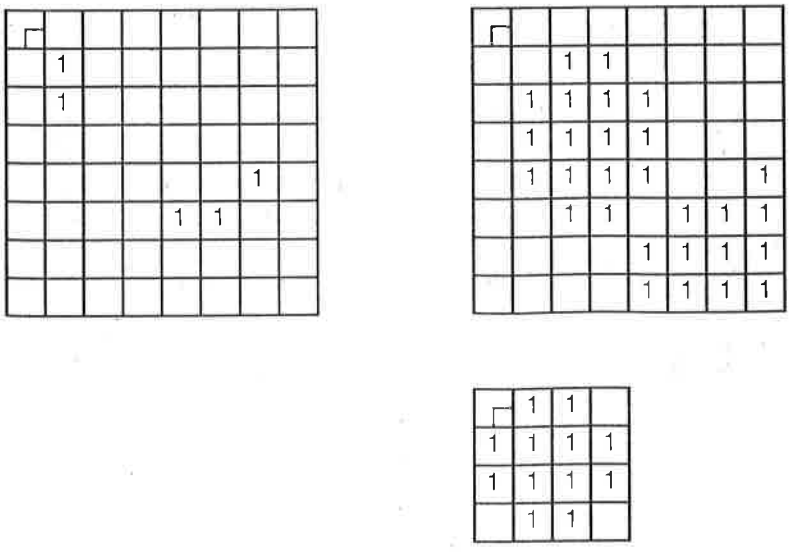

Figure B–8. Dilation by an octagonal structuring element.

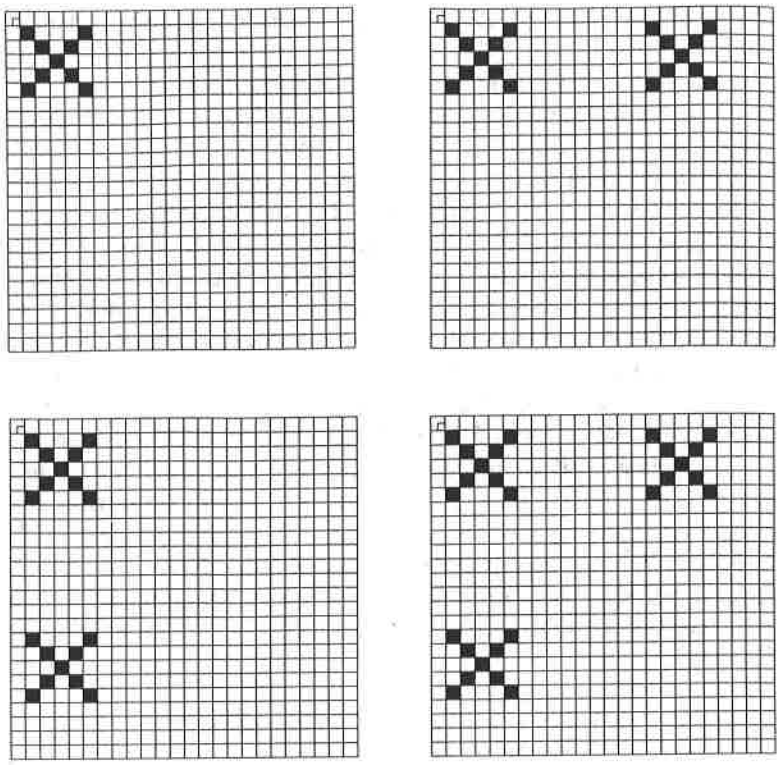

Figure B–9. Dilation with an additional structuring element.

That is, dilation is associative. The form $(A \oplus B) \oplus C$ gives a considerable savings in number of operations to be performed when A is the image and $B \oplus C$ is the structuring element. The savings come about because a brute force dilation by $B \oplus C$ might take as many as N^2 operations, whereas first dilating A by B and then dilating the result by C could take as few as $2N$ operations, where N is the number of elements in B and in C.

The dilation of A by B can be computed as the union of translations of A by the elements of B. That is,

$$A \oplus \mathrm{B} = \bigcup_{b \in B} (A)_b$$

Erosion

Erosion is the morphological dual to dilation. It is normally used to eliminate small protrusions on a shape or islands in an image. It can

widen cracks and holes. Erosion combines two sets using vector subtraction of set elements. If A and B are sets in Euclidean N-space, the *erosion* of A by B is the set of all elements x for which $x + b \in A$ for every $b \in B$.

Let us denote the erosion of A by B as $A \ominus B$. Erosion is thus defined by

$$A \ominus B = \{x \in E^N \mid x + b \in A \quad \text{for every } b \in B\}$$

The utility of the erosion transformation is better appreciated when the erosion is expressed in a different form (that given by Matheron, 1975). The erosion of an image A by a structuring element B is the set of all elements x of E^N for which B translated to x is contained in A.

$$A \ominus B = \{x \in E^N \mid (B)_x \subseteq A\}$$

Erosion is illustrated in Figure B–10. The upper left shows the input image consisting of two blobs. The upper right shows the input image eroded by the structuring element

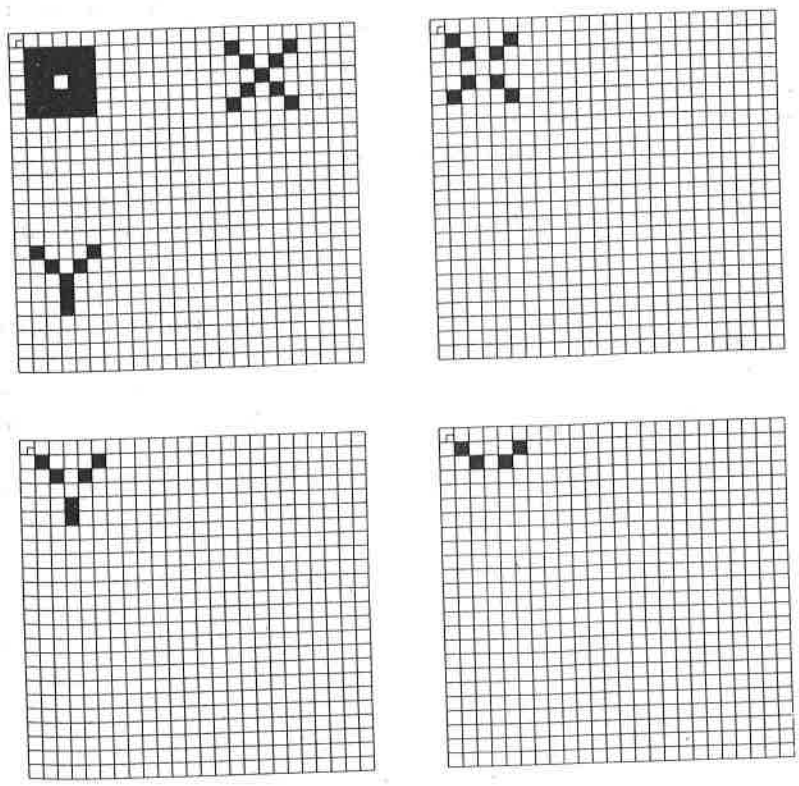

Figure B–10. Erosion of an image of two blobs.

$$\{(0, 0), (-14, 0)\}$$

The lower left shows the input image eroded by the structuring element

$$\{(0, 0), (0, -14)\}$$

The lower right shows the input image eroded by the structuring element

$$\{(0, 0), (0, -14), (-14, 0)\}$$

Openings and Closings

In practice, dilations and erosions are usually employed in pairs, either dilation of an image followed by the erosion of the dilated result or image erosion followed by dilation. In either case, the result of iteratively applied dilations and erosions is an elimination of specific image detail smaller than the structuring element without the global geometric distortion of unsuppressed features. The *opening* of image B by structuring element K is denoted by $B \circ K$ and is defined as $B \circ K = (B \ominus K) \oplus K$. The *closing* of image B by structuring element K is denoted by $B \bullet K$ and is defined by $B \bullet K = (B \oplus K) \ominus K$.

For example, opening an image with a disk-shaped structuring element smooths the contour, breaks narrow isthmuses, and eliminates small islands and sharp peaks or capes. Closing an image with a disk-structuring element smooths the contours, fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps on the contours.

Of particular significance is the fact that image transformations employing iteratively applied dilations and erosions are idempotent, that is, their reapplication effects no further changes to the previously transformed result. The practical importance of idempotent transformations is that they comprise complete and closed stages of image analysis algorithms because shapes can be naturally described in terms of under what structuring elements they can be opened or can be closed and yet remain the same.

If B is unchanged by opening it with K, we say that B is open with respect to K, whereas if B is unchanged by closing it with K, then B is closed with respect to K.

Sets that can be expressed as some set dilated by K are necessarily open under K.

$$A \oplus K = (A \oplus K) \circ K$$

Similarly, images that have been eroded by K are necessarily closed under K.

$$A \ominus K = (A \ominus K) \bullet K$$

From these two facts, the idempotency of opening and closing follows. Openings and closings have other properties. For example, it follows immediately from the increasing property of dilation and the increasing property of erosion that both opening and closing are increasing.

There is a nice geometric characterization to the opening operation. This characterization justifies why mathematical morphology provides material for extracting shape information from image data. The opening of A by B is the union of all translations of B that are contained in A.

Discussion

Dilation, erosion, opening, and closing can be used as the basis of image algebras. These algebras allow the definition of shape transformations that are customized for particular applications. A sequence of these operations, with suitable structuring elements, can be used to identify gear teeth in images of gears, or holes of particular sizes in images of machine parts. These techniques have been successfully applied to the problem of visually detecting shorts and open circuits in the wiring of printed circuit boards. This is illustrated schematically in Figure B–11.

Opening removes small protrusions, isthmuses and islands. Closing removes small cracks, bays, and holes. Taking the exclusive-OR of the resulting image with the original gives an image in which only potential defects remain. The original binary image is shown in the upper left. The result after erosion is in the upper center. After dilating that image, the result in the upper right is obtained. A second step of dilation takes us to the result in the lower left, and then another erosion takes us to the lower center. Exclusive-ORing this with the original produces the image of the isolated defects, shown in the lower right.

These operations can be efficiently computed with appropriate hardware. An entire session of the 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management was devoted to computer architecture specialized to perform morphological operations. Papers included those by McCubbrey and Lougheed (1985), Wilson (1985), Kimmel, Jaffe, Manderville, and Lavin (1985), Leonard (1985), Pratt (1985), and Haralick (1985). Gerritsen and Verbeek (1984) show how convolution followed by a table lookup operation can accomplish binary morphology operations.

Mathematical morphology is being extended to encompass more and more general classes of operators. Gray-scale extensions have been studied. Efforts have been made to cast morphology operations into a digital signal processing framework. A tutorial article presenting many more of the details of mathematical morphology is the paper by Haralick, Sternberg, and Zhuang (1987).

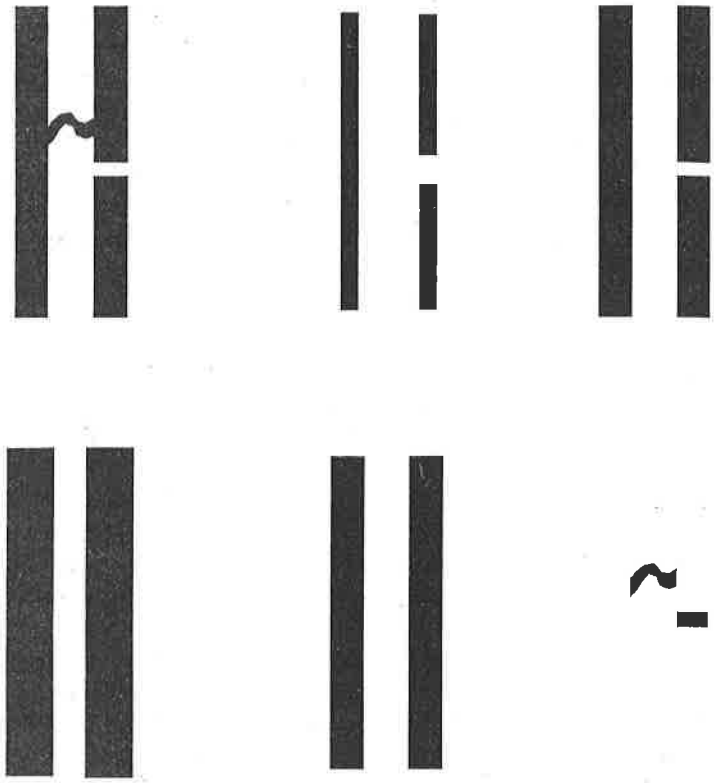

Figure B–11. Application of opening and closing to PC board inspection.

C. COMPUTATIONAL VISION ADVANCES

C1. Shape Representation and Analysis

THE TASK FACING a computational vision system is to compute descriptions of a 3-D scene given projections of that scene into 2-D images. The current paradigm for computational vision research assumes that a system must be structured into levels or modules with various special-purpose representations at each level and that processes transform descriptions from one representation into another. Each representation serves to make explicit some properties of the image or scene and leave others implicit. The choice of a representation for each particular level constitutes the determining design decisions for a particular vision system. A wide variety of criteria enter into these choices; it is important to discover and explicate these criteria. (See Articles II.C5 and XIII.D5, 6 for discussions of earlier work.)

For vision, we can distinguish four varieties of domains that need explicit shape representation:

1. *Functions of one variable* such as those that occur in, say, examining the intensity profile across a discontinuity in an image.
2. *2-D shapes* such as the contour of an image region.
3. *Functions of two variables* such as the depth map of a visible surface that gives depth as a function of x and y.
4. *3-D shapes* such as the bounding surface of a solid object.

This section will be structured around descriptions of some advances in representation techniques for each of these domains.

C2. Criteria for Shape Representation

GIVEN THAT THE CONCEPT of "shape" is intuitive rather than formal and the fact that for any shape domain there are infinitely many possible representations of the "shape" of an object, many researchers have felt the need to explicate *adequacy criteria* for shape representations. These necessary criteria allow us to make sensible design decisions and trade-offs when choosing a "good" shape representation. Here we shall provide

a set of criteria based on the current state of the art (Marr and Nishihara, 1978; Binford, 1982; Brady, 1983; Mokhtarian and Mackworth, 1986a; Mackworth, 1987; Woodham, 1987a, 1987b).

Computable: Given the input data and model assumptions, the representation should be efficiently computable on a suitable serial or parallel architecture; that is, the computational complexity should be a low-order polynomial in time, space, and number of processors.

Local: A useful representation must still be computable for portions of an object. If the parts of the representation depend only on data in a defined neighborhood of the object, it has local support. If only some of the neighborhoods are present in the data, a useful representation can still be computed for occluded or distorted objects. Also the inherent parallelism can be exploited by using special-purpose architectures that process the neighborhoods in parallel.

Stable: A small local change in the object should induce a small local change in the representation. This is required for noise resistance and shape matching.

Unique: A given object must have a unique representation. The mapping from object to representation must be a single-valued function from the object domain to the representation domain. This rules out schemes that make arbitrary choices about the mapping.

Complete: For a large and important domain of objects, the function from object to representation should be "total"; that is, for each and every object there is a corresponding representation.

Invertible: Ideally the mapping from object to representation should be invertible (also called *rich* or *information preserving*). If the object-to-representation mapping is many-to-one, different objects cannot be distinguished on the basis of their representation. Thus the mapping must be one-to-one; that is, a representation specifies a unique object. If the one-to-one mapping is computationally invertible (which it might not be even if the mathematical mapping is one-to-one), then, for example, the visual appearance of an object can be predicted from its representations.

Invariant: If a pair of 2-D or 3-D objects differ only by a rigid translation or rotation or by a magnification (a uniform change in scale), we say they have the same shape. Accordingly we require that the shape representation be *essentially* invariant under these transformations. This requirement is apparently in conflict with the requirement for invertibility; two objects seem to have the same representation. However, if this representation includes translation, rotation, and magnification parameters as components, the conflict with invertibility is resolved.

Scale-sensitive: The representation should incorporate information about the object at varying levels of detail, coarse to fine. This usually

corresponds to varying the size of the "neighborhood of local support." It also contributes to the required stability and matching properties of the representation. By suppressing the fine detail in the representation, we can concentrate on the broad, overall shape features and save on storage and processing time at the expense of accuracy and the invertibility criterion.

Composite: 2-D and 3-D objects have a natural recursive part-whole composition structure that should be explicit in the representation.

Matchable: The representation should be designed to support a matching process that compares two shape descriptions (one, for example, from the image; the other, a stored prototype) and returns a description of their difference. This includes computing properties of an object using the representation. For example, we can determine whether or not an object is symmetric by matching its description with that of a generic symmetric object.

Generic: A shape representation should support the description of a generic class of objects as well as specific objects (perhaps through parameterization). Thus if the representation is invertible as well as generic, it can be used in symbolically predicting appearances.

Refinable: If the representation supports generic descriptions, they should be refinable with the acquisition of more constraints (from the image or elsewhere) to characterize a more specific object class.

These dozen criteria serve as useful tools not only for the evaluation of existing shape representations, but also for their elaboration and the discovery of new methods. We now turn to examine their applications to the four levels of object domains found in most vision systems.

Descriptions of Functions of One Variable

Suppose we wish to describe a noisy one-dimensional signal $f(x)$ in order, say, to find intensity changes. A Fourier decomposition of the signal has many desirable properties. It satisfies many of our criteria, but crucially fails to satisfy the criterion of locality: that each of the Fourier basis functions have an infinite neighborhood of support.

Suppose we want to use the description to find edges characterized by abrupt changes of intensity. If the signal has undergone significant degradation due to blurring and noise processes, an edge can be said to exist at location x and scale σ if the slope at point x and scale σ achieves a local maximum with respect to x. To make this precise, the slope at point x and scale σ can be defined to be the result of differentiating the function $F(x, \sigma)$ that arises from convolving $f(x)$ with the Gaussian $G(x, \sigma)$.

$$F(x, \sigma) = G(x, \sigma) \otimes f(x)$$
$$= \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-u)^2/2\sigma^2} f(u)du$$

An "edge" exists at location x and σ wherever $F_x(x, \sigma)$ reaches a maximum or minimum or where

$$F_{xx}(x, \sigma) = 0 \quad \text{and} \quad F_{xxx}(x, \sigma) \neq 0$$

This technique, introduced by Stansfield (1980) and, most effectively, by Witkin (1983), is known as *scale-space filtering*. It plays an important role in many new techniques for shape representation. The (x, σ) space, known as scale space, can be used to represent a binary image, the scale space image of $f(x)$, with a mark wherever $F_{xx}(x, \sigma) = 0$ and $F_{xxx} \neq 0$.

We note the following:

$$F_{xx}(x, \sigma) = \frac{\partial^2}{\partial x^2} [G(x, \sigma) \otimes f(x)]$$
$$= G_{xx}(x, \sigma) \otimes f(x)$$

Thus the scale space image can be computed by precomputing the masks, $G_{xx}(x, \sigma)$.

For an extensive discussion of scale-space methods with good examples, see Witkin (1987). The Gaussian is the only filter that does not create generic zero-crossings as the scale increases, and this is true in any dimension (Babaud et al., 1986, Yuille and Poggio, 1986). This key *monotonic property* means that the scale-space image of a function of one variable is hierarchically structured. In the scale-space image, the contours of $F_{xx}(x, \sigma) = 0$ only have maxima—not minima. This property allowed Witkin (1983) to define the interval tree in scale space. The "edges" whose scales exceed any given value of σ partition the x-axis into intervals. As σ is decreased from a coarse scale, new edges appear in pairs dividing the containing intervals into three subintervals.

This subdivision process continues as σ is decreased down to the finest available scale (Witkin, 1987). This *interval tree* can be used as a representation of the shape of the function that satisfies many of the criteria of Section C2. It is not as stable as one might like—small changes in the function can produce large changes in the topology of the interval tree. Surprisingly, it is invertible.

Yuille and Poggio (1984) show that the scale-space image uniquely characterizes the curve modulo a multiplicative constant and a linear additive component, but the inversion may not be "computationally well conditioned" even if the slope or strength of each zero-crossing is known (Hummel, 1986).

Mokhtarian and Mackworth (1986) show how to match scale-space images using the A* algorithm (cf. Volume I).

Clark (1987) observes that the "edges" marked in the scale-space image can be classified as "authentic edges" and "phantom edges." "Authentic edges" correspond to positive maxima and negative minima of $F_x(x, \sigma)$, whereas "phantom edges" correspond to negative maxima and positive minima of $F_x(x, \sigma)$. These can be simply discriminated based on the sign of $F_x(x, \sigma)\ F_{xxx}(x, \sigma)$. The removal of the phantom edges from the scale-space image produces a reduced scale-space image that is not as well behaved as the scale-space image.

Canny (1986) presents an edge detector that is almost optimum with respect to the tradeoff of detectability in the presence of noise and localization based on similar multiscale techniques. His operator detects local maxima in the convolution with the first derivative of a Gaussian. Deriche (1987) improves on Canny's results.

Descriptions of Two-Dimensional Shapes

An arbitrary curve in 2-D space is the simplest generalization possible beyond a function of one variable. A 2-D connected region in a binary image may be represented by the simple closed curve corresponding to the exterior boundary and zero or more closed curves corresponding to the boundaries of any holes. It is, therefore, important to have shape representations for open and closed curves that satisfy our criteria.

Many current vision systems use global 2-D shape-dependent features such as the number of holes, aspect ratio, the ratio of perimeter squared to the area, moments of inertia, and the like (Brady, 1983). Although such properties can be computed efficiently and can be used in simple industrial inspection jobs (where the lighting can be controlled and the context is narrowly limited), they are not sufficiently local, stable, invertible, scale-sensitive, composite, generic, or refinable to handle more general vision tasks such as interpreting outdoor scenes.

Brady and Asada (1984) proposed smoothed local symmetries as a representation of 2-D shape. Essentially a local symmetry exists for a pair of points A and B on a simple smooth closed curve if and only if the right bisector of the straight line joining A and B serves as an axis of symmetry for the tangents to the curve at A and B.

In Figure C–1 the point O lies on an axis of local symmetry. In theory, for all pairs of points on the curve, we compute the set of all points that lie on axes of local symmetry. Then we compute the maximal smooth loci of those points. Each locus is a candidate axis. A local symmetry constitutes a locally plausible way to describe a portion of the contour and the region it subtends, called the "cover" of that axis. Each axis whose cover is properly contained in the cover of another axis is deleted to give the final representation.

In practice, the algorithm must contend with incomplete and noisy

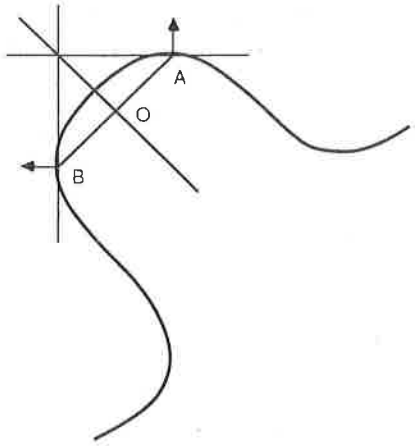

Figure C–1. Point O lies on an axis of local symmetry.

data and so is more complicated and must be optimized for better efficiency. Brady and Asada (1984) propose placing *knots* at points of high curvature on the bounding contour, constructing a piecewise smooth approximation to the curve using straight lines and circles, and then computing the smoothed local symmetries of the approximation to the contour. Asada and Brady (1986) propose using the scale-space image of the curvature as a function of arc length to recognize the existence of certain primitives that embody orientation and curvature discontinuities.

The smoothed local symmetries representation is a development of the symmetric axis transform (Blum and Nagel, 1978) and the 3-D generalized cylinder representation (see Section C3). The symmetric axis transform is the locus of the centers of maximal circles contained within the region. Such circles must touch the boundary at two points, at least.

Hoffman and Richards (1982) also used curvature, proposing that knots be placed at negative minima of curvature and that a dictionary of "codons" be used as primitives between the knots. This has the advantage of nicely explaining figure-ground reversal segmentation phenomena as occur in Rubin's Vase (see Figure C–2), for example, but it does not satisfy the need for scale-sensitive and stable representations.

In searching for ways to generalize the scale-space transform from functions of one variable to two-dimensional shape analysis, several approaches are possible. We have already mentioned smoothing the boundary curvature as a function of arc length (Asada and Brady, 1986). Another approach would be to smooth the 2-D image of the region with a 2-D Gaussian filter and extract the zero-crossings of the Laplacian

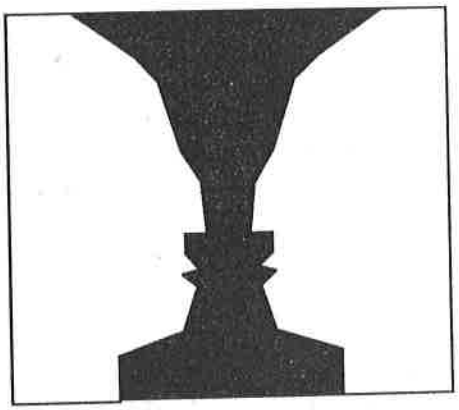

Figure C–2. Rubin's Vase.

operator $(\partial^2/\partial x^2 + \partial^2/\partial y^2)$ (Marr, 1982). Unfortunately, as Yuille and Poggio (1986) show, as the scale of the filter is increased, a zero-crossing contour so obtained can split into two, or two contours can merge into one. Babaud et al. (1986) show a dumb-bell–shaped region that exhibits both these behaviors. The single initial contour splits into two as the scale increases, but as it increases still further, they merge back again into a single contour. Although the monotonic property discussed earlier holds for 2-D smoothing in the sense that no new contour can appear (without splitting off an existing contour), this behavior is nonmonotonic in the sense that the number of regions defined goes from 1 to 2 and back to 1 as σ increases. This is not satisfactory from the point of view of scale-sensitivity.

Accordingly, Mokhtarian and Mackworth (1986) propose a boundary smoothing approach. It is not appropriate to smooth $y = y(x)$ as a function of x for several reasons, one of which is simply that a smoothed version of the curve and the smoothed version of the curve rotated through $\pi/2$ would have radically different shapes, which violates our invariance criterion. They propose smoothing in a natural path-based coordinate frame. The curve is described parametrically as

$$\{(x(t), y(t)) | t \in [0, 1)\}$$

where t is a linear function of path length. Then the curve is smoothed by a 1-D Gaussian kernel $G(t, \sigma)$. The resultant smoothed curve represents the original curve at coarser detail. If the original curve is closed, the smoothed curve is closed. The zeros of curvature (the inflection points) on the smoothed curves can be displayed as a map in (t, σ) space as a *generalized scale-space image* of the curve. This hierarchically structured

scale-space image is a useful representation of the shape of the curve or the region contained in the curve if it is closed. Mokhtarian and Mackworth (1986) show how to use this representation to match landforms in a map and a LANDSAT satellite image using a coarse-to-fine strategy. The major disadvantage of this representation is that all simple convex curves have the same representation, the empty scale-space image, because they have no points of inflection. Compared with the alternative of smoothing the curvature function (Asada and Brady, 1986), it has the advantage of preserving the closure of closed curves (Horn and Weldon, 1986).

Horn and Weldon (1986) propose a representation called the *extended circular image* for simple, closed convex curves. This is the 2-D analog of the *extended Gaussian image* representation for convex 3-D objects (both discussed later in this article). In the extended circular image, we are given the radius of curvature R as a function of normal direction ψ. For a circle radius R, we have $R(\psi) = R$. In general, $R(\psi) = 1/\kappa(s)$, where $\kappa(s)$ is the curvature as a function of path length. The integral of the extended circular image over a range of angles is the length of the portion of the curve with a normal direction within that range. The extended circular image of a closed convex curve is unique and invertible. One may smooth a closed convex curve by convolving its extended circular image with a smoothing filter (such as the Gaussian) and inverting the result to produce a smoothed, closed convex curve. This representation has most of the properties of a good shape representation. With regard to completeness, its domain is complementary to that of Mokhtarian and Mackworth with respect to the set of all closed curves.

Descriptions of Functions of Two Variables

It is important to have good shape descriptions of single-valued functions of two variables—surfaces in 3-space. Describing the image intensity surface $I(x, y)$ and the visible surface depth map $z(x, y)$ are two examples of where this is needed. The depth map $z(x, y)$ may be an intermediate stage of description of the scene or it may be obtained directly as a *range image* from an active sensor using sonar or structured light from, say, a laser. Besl and Jain (1985) survey some recent work in the description of surfaces.

Haralick et al. (1983) survey several papers on topographical classification of digital surface features and propose a descriptive scheme based on a set of ten labels that include features such as peak, ridge, saddle, planar, and pit. At each pixel in the intensity image, the parameters of an analytical *facet model* are estimated to give the best local fit. Those parameters can then be used to determine slope, the principal directions of curvature, and the two principal curvatures that determine

the pixel labels. Nackman (1984) proposes a similar scheme for segmenting surfaces based on critical points (local maxima, local minima, and saddle points).

Scale-sensitive descriptions of functions of two variables may be obtained using the 2-D scale-space approach with the drawbacks discussed in the previous section.

Recovery of the depth map $z(x, y)$ from the image intensity function $I(x, y)$ is in general an ill-posed problem (Tikhonov and Arsenin, 1977) because the solution is not unique—the imaging process is not uniquely invertible. If, however, the imaging model is simple, we can recover the "best" surface that could have produced the given image, under known illumination and imaging geometry and radiometry conditions. Suppose the image intensity at a point on the surface imaged is known to depend only on the surface gradient (p, q) (Mackworth, 1983). Then the image irradiance equation takes the form (Horn, 1986):

$$I(x, y) = R(p, q)$$

Clearly, given $I(x, y)$ on a digital grid as $I_{ij} = I(x_i, y_i)$ and the functional form of $R(p, q)$, we cannot determine $\{p_{ij}\}$ and $\{q_{ij}\}$ because they are underconstrained. The extra constraint necessary can be provided by insisting that the surface found be the one that minimizes a weighted sum of the squared error in the image irradiance equation and a quadratic measure of the smoothness of the surface (Ikeuchi and Horn, 1981). This functional essentially selects a single surface from the set of all possible surfaces. We can determine this surface by setting the partial derivatives of this functional with respect to the orientation parameters equal to zero and solving the large sparse set of linear equations by an iterative relaxation method. Terzopoulos (1986, 1987) has shown that the convergence of this process can be accelerated using multigrid relaxation methods, again demonstrating the importance of scale-sensitive descriptions. (Further details on multigrid methods are given in Section D.) Woodham (1987a) and Horn (1986) provide excellent overviews of the shape-from-shading method and the use of multiple light sources and photometric stereo (Woodham, 1980) to overconstrain the surface gradients.

Another approach to determining shape-from-X is based on fractal modeling of the surface (Pentland, 1983). If we assume that a surface has isotropic fractal characteristics, then under certain imaging assumptions the image intensity surface will also have fractal characteristics. By measuring those characteristics, we can arrive at estimates of the 3-D surface characteristics.

Another class of shape-from-X methods is shape-from-contour. The blocks world scene domain was the development ground for many of these methods as documented in Volume III of the *Handbook*. Despite the

fact that most researchers abandoned the blocks world, many important theoretical and practical problems remained unsolved. Sugihara (1986) has continued to attack those problems. His book is an excellent summary of his results. It is organized around the exposition of a four-module procedure for the interpretation of line drawings as polyhedral scenes. The first module is the classical Huffman-Clowes labeling. However, Huffman-Clowes labeling may generate specious labelings that are not realizable as polyhedral scenes (Mackworth, 1977a). Accordingly, the second module determines which of the proposed labelings are realizable. This test of geometric feasibility is carried out by reducing the problem to the determination of a feasible solution to a linear programming problem. The third module allows tolerance in the definition of geometric feasibility by removing redundant constraints and "correcting" the original line drawing (moving vertices and the like). The fourth module allows the use of additional information sources such as 3-D range finding, surface shading, and texture to pick a unique scene interpretation from the infinite number of possible scenes in the equivalence class of interpretations depicted by the image.

There were at least four main open issues at the end of the first decade of blocks world research. First, procedures such as Huffman-Clowes-Waltz labeling and gradient space reasoning applied necessary but not sufficient tests for realizability (see Article XIII.B5, Vol. III). Second, it was not clear how to characterize the degrees of freedom in the scene equivalence class. Third, the computational complexity of the problems and their algorithms was not understood (see Section C4 on constraint satisfaction). Fourth, it was not known how to apply these methods to "real" images, integrating these methods with other shape-from-X methods and coping with noisy data. Sugihara has contributed substantially to the solution of each of these problems.

The old question as to whether this approach will generalize outside the blocks world must be faced. Obviously the techniques will not work on images of tea cups or clouds. But, just as with the gradient space approach, the underlying methodology does apply generally. It is a demonstration of the power of characterizing the equivalence class of scenes in terms of the constraints from the image imposed on the a priori degrees of freedom of the scene (Mackworth, 1983) and furthermore of finding a unique scene by minimizing a functional over the equivalence class. Moreover, Sugihara (1986) contributes to the theme of structure rigidity by developing the analogy between the duality principle behind gradient space structures and the corresponding duality principle behind force diagrams of rod and pin structures. These ideas will generalize far beyond the blocks world as we design and build large space frame structures on earth and beyond. The developments in Sugihara's book depend on recent advances in matroid theory, which makes them somewhat inaccessible to many readers.

Representing Three-Dimensional Shapes

As discussed in Article XIII.D6 of Volume III, three dimensional object shape representations can be vertex and edge-oriented, surface-oriented, or volume-oriented. The volume-oriented representation most commonly mentioned in vision work is the generalized cylinder. As Binford (1982) demonstrates, it satisfies many of our shape representation criteria. But note that it may not be unique—often an arbitrary choice of axis must be made. Shafer and Kanade (1983) provide a very useful categorization of generalized cones and cylinders. Brady and Asada's (1984) smoothed local symmetries can be seen as a 2-D generalized cone representation. It may be useful for accessing stored 3-D representations.

The extended Gaussian image (EGI) is an important 3-D object representation tool that has received attention recently (Horn, 1986, Chapter 16). For a convex solid object, this is an invertible shape representation. It can be derived from the needle map produced by photometric stereo or a depth map produced either by binocular stereo or by a direct depth sensor.

The extended Gaussian image of an object is defined on the unit Gaussian sphere that corresponds to the set of all possible normals on the object. For a polyhedron, the EGI is a set of impulses. Each impulse corresponds to the face with the appropriate normal. The weight of the impulse is the area of the face. Little (1983) shows how to invert the representation; he reports on an iterative algorithm that reconstructs a polyhedron, given the areas and orientations of the faces.

The EGI of a smooth convex object must be approximated by tessellating the Gaussian sphere. The weight of each facet on the sphere equals the surface area on the object for which the normals lie within the facet. Coarse-to-fine EGI representations can be constructed using successively finer tessellations. The extended circular image discussed in Section A4 is the 2-D analog of the EGI.

An interesting proposal for 3-D shape modeling is the superquadratics approach (Barr, 1981) advanced by Pentland (1986). This can be seen as an attractive alternative to the generalized cones approach. The primitives in this approach are chosen from a parameterized family of superquadric 3-D shapes. A simple superquadric is a shape described by the following equations:

$$\begin{aligned} x(\eta, \omega) &= (\cos \eta)^{\epsilon_1} (\cos \omega)^{\epsilon_2} \\ y(\eta, \omega) &= (\cos \eta)^{\epsilon_1} (\sin \omega)^{\epsilon_2} \\ z(\eta, \omega) &= (\sin \omega)^{\epsilon_2} \end{aligned}$$

where $(x(\eta, \omega), y(\eta, \omega), z(\eta, \omega))$ is a 3-D vector that sweeps out a surface parameterized by latitude η and longitude ω. The shape of the surface is controlled by the parameters ϵ_1, and ϵ_2. For example, if $\epsilon_1 = \epsilon_2 = 1$, the shape is a sphere. But the complete family of superquadrics also

includes cubes, diamonds, pyramids, and cylinders. The complete modeling system allows these parts to be stretched, bent, twisted, and tapered, and then combined using Boolean combinations (ANDs, ORs, and NOTs) to form new prototypes that can then, recursively, be again deformed and combined with other prototypes. From the perspective of our adequacy criteria for shape representation, this proposal offers several advantages. The completeness of the domain of coverage is high, though there are still some difficulties with such things as pentagonal solids. Most other proposals such as generalized cones are essentially subsumed by superquadrics.

C3. Object Recognition

LET US NOW REVIEW developments in the automatic recognition of objects. We refer to several important systems that focus on recognition in a "bin-picking" (factory robotics) environment. Bolles and Cain (1982) present the "local-feature-focus method" for recognizing and located 2-D possibly occluded objects. An object model consists of a graph whose vertices represent features such as corners and holes and whose edges are labeled with the distance and relative orientation of the two features related by the edge. The edge constraints help to control the matching process.

Similarly, Grimson and Lozano-Perez (1984) show how to use local measurements of 3-D positions and surface normals to identify and locate objects in a scene from a set of known objects. The objects are modeled as polyhedra with three degrees of rotational freedom and three degrees of translational freedom. The local measurements could come from a set of tactile sensors or 3-D range sensors. The measurements are assumed to have a small range of possible errors, although the normal measurements are assumed to be less reliable than the position measurements. The problem can easily be formulated as an exhaustive search problem; the trick is to reduce the search space to one of manageable size. For each object in the repertoire, the system searches an *interpretation tree*. If there are s measurements and n faces for the object, the tree has s levels and a branching factor of n at each level. The tree has, potentially, n^s leaves each corresponding to a set of assignments of the s measurements to the n faces. However, the search tree may be cut off above the leaf level using such binary constraints as:

1. *Distance constraint*—The distance between a pair of measurements must be a possible distance between the pair of faces assigned to them.

2. *Angle constraint*—The range of angles between measured normals must include the angle between the pair of faces assigned to them.

These and other constraints are used very effectively to prune the interpretation trees as early as possible. Considerable data is provided to demonstrate the power of using such local constraints to control the global object matching process.

Grimson (1986) provides careful combinatorial analysis of the efficacy of various constraints. Grimson (1987) extends the approach to the recognition of objects that can vary in parameterized ways, with parts that may have rotational, translational, scaling, and stretching degrees of freedom.

Bolles and Horaud (1986) describe 3DPO, a system for determining the 3-D position and location of parts in a jumbled bin of identical parts. It generates hypotheses about part locations using 3-D edge features extracted from range data and then matches distinctive features to confirm or refine the match.

Another successful approach to the bin-picking problem has been reported by Ikeuchi and Horn (1984) (see also Horn (1986), Chapter 18). This approach does not use direct range data, relying instead on photometric stereo (Woodham, 1980, and Article XIII.A, Vol. III). Multiple images are captured, changing the position of the light source but keeping the camera in place. These images, combined with a reflectance map model of the imaging situation and the surface reflectance, provide sufficient constraints to extract an image-registered map of surface gradients (the needle diagram).

The needle diagram can then be mapped onto a tessellated Gaussian sphere, giving an orientation histogram where each facet contains the sum of the object surface areas corresponding to that range of orientations. This is a discrete approximation to the visible half of the extended Gaussian image (Section C2). This shape representation can be matched against a stored histogram obtained from a prototype model of the part. The best match gives the attitude (but not the distance away) of the object to be picked up. The object gripper is moved out along the ray from the camera on which the object is known to lie until a proximity sensor is triggered, at which point the gripper can be oriented for the known attitude of the object. The object is then grasped and removed.

Advances in 3-D object recognition from a single intensity image beyond ACRONYM (Brooks, 1981) (*Handbook*, Article XIII.F3, Vol. III) have been reported by Goad (1983) and Lowe (1985, 1987). Goad presents an interesting view of recognition as special-purpose automatic programming. His assumption is that a program to recognize a particular object can be optimized offline for that object by considering all possible views of the object to minimize actual recognition time. Recognition is seen as

a process of searching for the camera viewpoint in an object-centered coordinate system. As with Grimson and Lozano-Perez (1987), this process is seen as a tree search that matches image data to model data. In this case the matched data are lines from the image with edges in the model. The order of matching is precomputed to minimize search times. Recognition times on the order of one second are reported on a 1 MIP machine.

Lowe (1987) has also described a system, SCERPO, that can recognize and locate 3-D objects in single gray-scale images. The system first extracts edge-based features and forms perceptual groups (based on collinearity, parallelism, and proximity) that are likely to be invariant over a wide range of viewpoints. These are then matched against object structures with a probabilistic matching structure used to reduce the size of the search space. Finally, the unknown viewpoint and model parameters are determined by an iterative process of spatial correspondence based on Newton's method. Results on an image of a bin of disposable razors show robustness in the presence of occlusion and poor segmentation data.

Besl and Jain (1985) provide an extensive survey of 3-D object recognition systems and techniques with a particular emphasis on the use of range images.

C4. Constraint Satisfaction

THE TERM "constraint satisfaction" is used both to describe a class of problems and to name a method of solving these problems. Constraint satisfaction problems have considerable importance in vision and other areas of AI (Mackworth, 1987b). We shall briefly survey the two main approaches, emphasizing some recent results. Boolean constraint satisfaction problems, as typified by Huffman-Clowes-Waltz labeling, are one main class. The other is the class of optimization problems that used to be known as probabilistic relaxation problems.

Boolean Constraint Satisfaction Problems

A Boolean constraint satisfaction problem (CSP) is specified if we have a set of variables

$$V = \{v_1, v_2, \ldots, v_n\}$$

and a set of Boolean constraints limiting the set of allowed values for specified subsets of the variables. Each variable takes on values in some domain. The set of solutions to the CSP is the largest subset of the Cartesian product of the domains of the n variables such that each n-tuple in the set satisfies all the given constraint relations. We may have

to list or describe all the solutions, find one, or just report if the solution set contains any members—the decision problem (Mackworth, 1977b; Haralick and Shapiro, 1979).

For example, deciding if an image can be labeled using the Huffman-Clowes labels is a CSP decision problem. There the variables can correspond to the junctions, the domains to the set of possible corners allowed for each junction type, and the constraint relations to the binary constraint that the corners at each end of an edge must have the same label for the edge. Or, dually, we could set up a CSP with the variables corresponding to the edges, the domains to be the set of possible edge labels allowed, and the constraint relations to the k-ary relations corresponding to the set of possible corners allowed by each junction type.

Determining if a planar map can be colored with three colors is a Boolean CSP that is NP-complete; therefore efficient (polynomial) algorithms are unlikely to be found for the general class. Moreover, it has recently been shown that even the Huffman-Clowes labeling CSP is NP-complete (Kirousis and Papadimitriou, 1985).

Since the general problem may well require exponential time to solve, approaches have concentrated on polynomial *approximation algorithms* that enforce necessary but not sufficient conditions for the existence of a solution.

Waltz's (1975) filtering algorithm is one of the arc consistency approximation algorithms. These algorithms are members of a class of network consistency approximation algorithms (Mackworth, 1977b) further generalized by Freuder (1978). Mackworth and Freuder (1985) settled a long-standing debate by proving that Waltz's arc consistency algorithm requires time linear in the number of constraints, at most.

Although in the past it was felt that CSPs are amenable to parallel solution, Kasif (1986) showed that arc consistency is "inherently" a serial problem. Precisely, he has shown that arc consistency is log-space complete for P. (Log-space complete problems for P are those problems solvable on a single Turing machine in polynomial time.) The implication is that it is very unlikely that arc consistency can be solved in time polynomial in $\log n$ with a polynomial number of processors. This somewhat counterintuitive result can be understood better if we realize that we can set up CSPs with serial data dependencies. A local inconsistency can be discovered by one processor at a vertex, which when removed causes an inconsistency at an adjacent vertex and so on. Since this propagation is serial, all but one of the processes may be idle all the time.

Nudel (1983) has shown some tight results on expected time complexity for classes of CSP on a single processor. Mackworth, Mulder, and Havens (1985) have described a new algorithm, hierarchical arc consistency, that exploits the situation where the values within a domain can be organized hierarchically with common properties. They describe the application of the algorithm in a schema-based recognition system for

maps and provide theoretical and experimental complexity results. Malik (1987) describes the application of the hierarchical approach to line labeling.

Optimization Problems

In computational vision one is often not just satisfying a set of Boolean constraints, rather one is optimizing the degree to which the solution satisfies a variety of possibly conflicting constraints: trading one constraint off against another. For example, Zucker, Hummel, and Rosenfeld (1977 and as covered in Article XIII.E4) in a curve enhancement application attach weights or "probabilities" in [0, 1] to each of nine labels (corresponding to eight compass orientations and "no line") and the relation matrices or "compatibilities" have entries in [−1, 1] that measure the extent to which two values from related domains are compatible. This scheme, known as *probabilistic relaxation*, iterates the application of a parallel updating rule, modifying the weights in each domain until a fixed point is reached or some other stopping rule applies. For an excellent overview of applications of this paradigm, see the survey by Davis and Rosenfeld (1981).

The probabilistic interpretation has problems of semantics and convergence—other interpretations are now preferred (Ullman, 1979; Hummel and Zucker, 1983). Algorithms in this class have been called *cooperative algorithms* (Julesz, 1971; Marr, 1982). Compatible values in neighboring domains can cooperatively reinforce each other while incompatible values compete, trying to suppress each other. Each value in a domain is competing against each of the other values in that domain. Cooperative algorithms are attractive because they are inherently parallel, requiring only local neighborhood communication between uniform processors that need only simple arithmetic operations and limited memory such as is available on the Connection Machine (Hillis, 1985; and Article D5 of this Chapter). These features suggest implementations for lower level perception (such as stereo vision) in artificial and biological systems (Marr, 1982; Ikeuchi and Horn, 1981; Zucker, 1983; Ackley et al., 1985; Little et al., 1987).

The design of these algorithms is best based on the minimization of a figure-of-merit. Ikeuchi and Horn (1981), as described in Section C2, carry out shape-from-shading using a figure-of-merit based on a combination of a measure of deviation from the image data and a measure of surface smoothness. The iterative relaxation solution corresponds to using *gradient descent* on the figure-of-merit, searching for the best set of orientation values for the surface elements. Note that the domains do not consist of a finite set of values each with a weight in [0, 1], but rather they consist simply of one value that is the current best estimate of the local value of the solution.

Gradient descent techniques are only guaranteed to find the global minimum of the figure-of-merit or "energy" surface if that surface is everywhere an upward concave function of the state variables of the system. In that case, there is only one local minimum and it is the global minimum. If the surface has local minima that are not the global minimum, techniques such as "simulated annealing" based on the Metropolis algorithm and the Boltzmann distribution can be used to escape local minima (Kirkpatrick et al., 1983, and Ackley et al., 1985).

Earlier, in Section C2, we described the shape-from-shading approach of Ikeuchi and Horn (1981) as an example of using regularization theory to solve an "ill-posed" problem. Regularization theory has been applied to a wide variety of early (low-level) vision problems (Poggio et al., 1985). For example, edge detection is an ill-posed problem because locating zeros of the numerical first derivative of the image is unstable; its solution does not depend continuously on the input intensities. Smoothing the image regularizes the problem, making discontinuity detection well posed (Hildreth, 1987).

Poggio, Voorhees, and Yuille (1984) and Torre and Poggio (1986) derive an optimal smoothing operator as follows. Suppose $I(x)$ is the image intensity function and $S(x)$ is the smoothed intensity function required. $S(x)$ should fit the image intensities closely and be as smooth as possible. In other words, $S(x)$ should minimize

$$\sum_{k=1}^{n} [I(x_k) - S(x_k)]^2 + \lambda \int |S''(x)|^2 dx$$

where λ is a constant controlling the tradeoff between fidelity to the image and smoothness. The solution to this minimization problem is equivalent to convolving the image with a cubic spline, which is similar to the Gaussian.

Hadamard defined a problem to be well posed if its solution exists, is unique, and depends continuously on the initial data. An ill-posed problem, one that is not well posed, fails to satisfy one or more of these conditions. A well-posed problem may, however, still be numerically ill conditioned and oversensitive to noise in the initial data (Poggio et al., 1985).

One general approach to the regularization of an ill-posed problem (Tikhonov and Arsenin, 1977) is as follows. Suppose we wish to solve the inverse problem: given $Az = y$, find z given the data y. This is solved by determining the function z that minimizes

$$\|Az - y\|^2 + \lambda \|Pz\|^2$$

where λ, the regularization parameter, controls the relative importance of the fit to the data and the degree of regularization of the solution. $\|Pz\|^2$ is the regularization criterion—usually some measure of "smoothness." Poggio, Torre, and Koch (1985) discuss the regularization of seven

ill-posed problems in early vision: edge detection, optical flow, surface reconstruction, spatiotemporal reconstruction, color, shape-from-shading, and stereo. Difficulties arise when the regularization imposes a smoothness constraint on the world that may be inappropriate. They also discuss how linear analog electrical and chemical neuron-based networks could solve the minimization problems that arise in a regularization approach.

D. VISION ARCHITECTURE

Overview

In the past decade the growth of interest in parallel computing within the computer vision community has been changing the field. More and more studies of machine vision are based on or motivated by a particular computer architecture. This section discusses the most influential architectural directions, along with their relationships with computer vision.

Architecture's Influence on Algorithms. Although much research in computer vision is driven purely by the insights about vision that the research community has accumulated, some research responds directly to the possibilities that new computer architectures offer. Computing with cellular-logic processors, connection machines, and real-time video processors has a flavor sufficiently different from conventional mainframe, mini, and micro computing that it has encouraged lines of research substantially different from those of the more traditional computational vision.

Those who have programmed highly parallel machines such as the CLIP4 and the Connection Machine say that after some experience, one begins to think "in parallel" on a whole new, higher, algorithmic plane than before. There are two reasons for this. First, the highly parallel machines offer relatively high-level instructions as the conceptual building blocks for algorithm design. A typical instruction of such a machine causes two images to be added together, whereas an ordinary computer could only add two individual pixels together in one instruction (or it might even take several instructions). Therefore the programmer is encouraged to work at a higher level of abstraction than otherwise. Second, these machines perform such operations very quickly—in a matter of microseconds, rather than seconds. This means that the programmer/researcher can effectively *interact* with the system at this high level of abstraction, rather than work with it in a batch mode.

Relationship with Data Structures. Some highly parallel computers are designed specifically to support operations on certain kinds of data structures. The CLIP4 operates on images. The Connection Machine can operate on images or pointer maps. Some pipelined systems such as Aspex's PIPE operate on video data streams. Parallel pyramid machines operate on pyramid data structures (see Uhr, 1987, for accounts of several pyramid machines).

By operating on these data structures as units, many of these parallel

architectures have an organizing principle built in; the data structure becomes the machine structure.

Parallelism in Vision

Although the computing community generally has been moving toward parallel processing, the case for parallelism in vision has been promoted with even greater strength. This is both because the human visual system seems to be a massively parallel system and because it is fairly obvious how images can be handled in regularly structured parallel systems (e.g., one processor per pixel). Nonetheless, parallelism can be used in vision in a significant variety of ways. A review of these will make the essential architectural alternatives clearer.

Parallel Methodologies

SIMD versus MIMD Systems. As is customary, let us divide the realm of parallel architectures into two broad groups:

1. Those in which a single program is being executed and in which at any one time all processors perform the same instruction on their own data.
2. Those in which processors follow different programs or different copies of the same program more or less independently on their own data.

In the terminology of Floyd, the first class of architectures are single-instruction-stream/multiple-data-stream (SIMD) systems, whereas the latter are multiple-instruction-stream/multiple-data-stream (MIMD) systems.

This distinction is a matter of processor autonomy; SIMD systems use many processing elements with little autonomy—they are permitted their own data but must execute programs in lockstep with one another. On the other hand, MIMD systems have highly autonomous processors that may work independently except when their programs call for communication and synchronization with other processors. In reality, many systems do not fall at one end or the other of this spectrum of processor autonomy; for example, their processors may have conditional instructions based on local conditions or they may have highly autonomous addressing capabilities. However, the SIMD-versus-MIMD distinction is very useful in examining the broad realm of parallel architectures.

In the vision community, there are vocal proponents of both SIMD and MIMD architectures. Consequently it is useful to understand the relative strengths and weaknesses of the two families.

First, we have the matter of cost. If cost were measured in the number

of logic gates in a computer, we could provide more processing elements in an SIMD system than in an MIMD system for the same cost because the SIMD system's processing elements do not require program counters and instruction-decoding logic. Proponents of MIMD systems argue that the flexibility of MIMD systems allows them to be manufactured and sold in larger quantities and therefore more cheaply than the more special-purpose SIMD systems.

Second, let us consider the programming problems these architectures present. The SIMD architectures tend to be structured according to some data structure such as a two-dimensional image array, and programming them is relatively easy. Whereas an MIMD system requires the programmer to write synchronization protocols and work out load-balancing arrangements, SIMD systems obviate most synchronization, and the programmer is not normally concerned with load balancing. This is because it is impractical to map computations onto the array in a fashion that does not follow the machine's special (e.g., image) structure.

In some ways, SIMD systems execute parallel computations more efficiently than MIMD systems—there is little communication overhead between processing elements because their interactions are preprogrammed and presynchronized. Depending on the interconnection network that links the processing elements, the overhead of routing data can be very low in SIMD systems. A limitation of SIMD systems, however, is that in computations where only one or a small number of processing elements are doing meaningful work, all the others must either operate on dummy or garbage data or wait idly. In MIMD systems, processors are not constrained by the architecture to idle if other meaningful tasks are ready to execute.

Data Flow. Another way of thinking about parallel processing is in terms of the flow of data through a network of operations where the data get transformed. The nodes of a data-flow network represent points in the process where the data is operated upon. It is not necessary that each node correspond to a processor; however, at some point during the computation, each node must be assigned to some processor so that the operation(s) can actually be performed. Several data objects might flow to the same node; one operation involving several operands might take place there, or a succession of operations might be performed at the node.

Data-flow paradigms have not been used much in machine vision except to the extent that image-stream processing may be thought of as data-flow processing. However, this particular kind of data-flow paradigm is better known as *pipelining*. In the future, general data-flow techniques may be appropriate for higher level (symbolic) processing of visual information.

MIMD Systems: Butterfly, Hypercubes, RP3, Warp. Computer systems that incorporate multiple processors, each executing an instruc-

tion sequence that is independent of the others, are of interest because of their ubiquity and flexibility, especially for vision-related computations at the symbolic level (rather than the pixel level). Several prominent MIMD systems are these: the Butterfly developed by Bolt, Beranek, and Newman, Inc., the Cosmic Cube developed at the California Institute of Technology (Seitz, 1984), the RP3 developed by IBM (Pfister et al., 1985), and the Warp at Carnegie-Mellon University. Of these, the two architectures designed principally for AI/vision applications are described here in more detail.

The Butterfly architecture covers a family of MIMD parallel processor systems that can have up to 256 processors in a system (Crowther et al., 1985). Each processor has a local memory with access time of about two microseconds, but the processor can also access the local memories of all the other processors through the network, and such an access takes approximately six microseconds. The Butterfly architecture works well on problems that can be decomposed for large-grain parallel processing with only modest amounts of interprocessor communication.

The Warp computer (Annaratone et al., 1987) is a linear array of programmable processors developed at Carnegie-Mellon University. Intended primarily for computer vision, it can also be applied to signal processing and scientific computation. A ten-processor prototype became operational in 1986. Originally it was conceived of as a "systolic" system in which data would be piped through the line of processors with SIMD control. Later it was decided to make the processors autonomous, and it became an MIMD system. The processors in the linear array operate on 32-bit words, and they are interconnected with 16-bit wide data paths. The linear array is connected through an interface unit to a host (Sun-3 workstation plus additional processors).

Multicomputers with Reconfigurable Interconnections. To avoid the limitations of any particular fixed interconnection structure, "reconfigurable" systems have been proposed. At a cost of slightly more switching hardware, the data and control paths among processing elements and control units can be made programmable. The CHiP computer and the PASM are two specific systems that have been described in the literature.

A CHiP (configurable highly parallel) computer is an array of processing elements interconnected with a system of wires and programmable switches (Snyder, 1982). Because the processing elements and switches are laid out on VLSI chips in an integrated manner, it is possible to achieve SIMD cellular array efficiency (including short data paths and synchronous communication). It is also possible to embed rich nonplanar interconnection graphs in a CHiP system because the switches can also be programmed to produce long, convoluted data paths containing crossovers.

The PASM (partitionable SIMD/MIMD) system permits the set of processing elements to be grouped (under program control) and each group associated with a separate control unit (Siegel et al., 1979 and Chu et al., 1987). The effect of this is to allow PASM to contain a multiplicity of SIMD parallel programs each executing independently of (or communicating asynchronously with) the others. A number of simulations have been reported that give the predicted performance of PASM on image analysis tasks.

Neighborhood Parallelism and Pipelined Systems. Another way to organize the processing of image data for parallel computation is to treat the *neighborhood* as the atomic unit of computation. In a neighborhood-parallel, pipelined image processing system, one neighborhood (generally a 3×3 set of pixels) is processed in a single machine cycle. The image data is shifted through the neighborhood processor so that every neighborhood (of the given size) is processed in a single scan of the image. Examples of neighborhood-parallel pipelined systems include PICAP (Kruse, 1980), the Cytocomputer (Lougheed et al., 1980), and PIPE (Kent et al., 1985), among others. It has also been proposed that such systems be implemented optically (Huang et al., 1987).

Let us describe PIPE (pipelined image processing engine) in more detail. It is a commercially available system that is oriented largely toward the processing of digitized video data in real time (30 frames/second). A PIPE consists of from three to eight "modular processing stages," each of which consists of a frame buffer, a neighborhood processing unit, and an address generator. In addition to these stages, there are an input stage, output stage, control unit, and control and data paths. Six modular processing stages and their interconnections are diagrammed in Figure D–1.

In typical operation, a stream of digitized video is passed from the input stage to the first processing stage, where a filtering operation is performed on it. By piping the image through the 3×3 neighborhood processor (which computes a single output value with the help of programmable lookup tables), the filtering is accomplished in a frame time. The result is then fed to the second stage where it is averaged with a similarly filtered picture from the video frame preceding the one on which this filtered image is based. This output is then passed to a third modular processing stage where an edge template is applied. The final outputs may be displayed or passed to a host for additional analysis. Because almost all aspects of the computation are programmable (neighborhood operators, data paths, and address generators), the programmer has substantial flexibility in designing algorithms for PIPE.

Although processing the nine points of a neighborhood in parallel can significantly speed up an image processing operation, an architecture that provides a separate processor for each pixel of an image can achieve

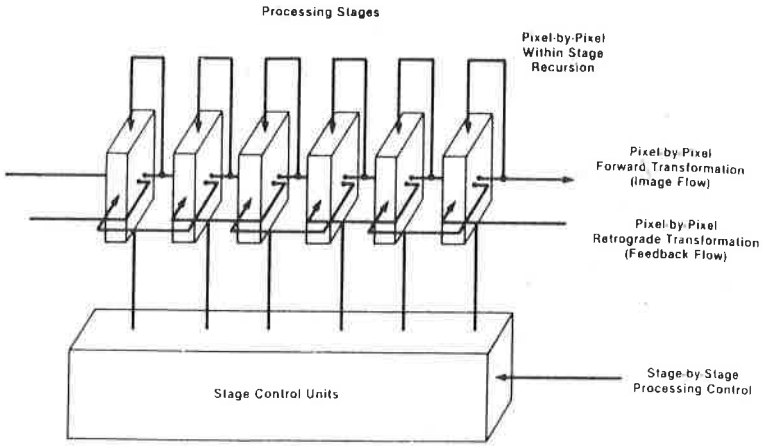

Figure D–1. Modular processing stages and data paths in PIPE (from Kent et al., 1985, courtesy of E. Kent).

much faster performance, albeit at an increased hardware cost. The mesh-based architectures of the next section demonstrate this.

Mesh Architectures

The period 1980 to 1987 saw major advances in the realization of massively parallel mesh-oriented processors. Notable systems in this group include the CLIP4, MPP, and the Connection Machine.

CLIP4. The first such machine, CLIP4 (Cellular Logic Image Processor, version 4), became operational in early 1980 at the Department of Physics and Astronomy, University College, London (Duff, 1976). The CLIP4 consists of a 96×96 array of processing elements controlled by a single program-interpretation unit. Each processing element (PE) of the CLIP4 has one bit of input from each of its eight nearest neighbors. These inputs can be masked under program control and then logically OR'ed and further combined with Boolean data from the PE's local memory. Thus each CLIP4 instruction performs a cellular-logic operation on an entire 96×96 binary image in one cycle. A conventional computer would have to perform over 10 billion operations per second to keep up with the CLIP4 (Preston and Duff, 1984).

MPP. The Massively Parallel Processor (MPP) became operational in 1983. Developed by Goodyear Aerospace under sponsorship of the NASA Goddard Space Flight Center, the MPP contains a 128×128 array of processing elements roughly comparable in power to the CLIP4 PEs.

Each PE in the MPP has a reconfigurable shift register that speeds up bit-serial arithmetic by a constant factor over the CLIP4; however, each PE in the MPP can only access a bit of data from one neighbor at a time, rather than eight at a time in CLIP4. The MPP augments the mesh with a "staging memory," which is provided to lessen the effect of the input/output bottleneck from which both the CLIP4 and the MPP suffer (Batcher, 1980). By using later technology than the CLIP4 and a larger array, the MPP achieves approximately the equivalent of one trillion operations per second on a conventional computer (Preston and Duff, 1984).

Multilevel Architectures

Mesh-based architectures are highly efficient for computing transformations of images where the output at a pixel is only a function of the local neighborhood of that pixel. However, many computer vision problems require the computation of more global and symbolic representations of an image. To make the more general kinds of computation efficient, meshes have been augmented in a variety of ways. The CLIP4 and MPP actually include a feature that lets the control unit know whether any PE has a nonzero value in its accumulator. However, this is a very minimal augmentation to a mesh.

Pyramid Machines. A relatively straightforward augmentation to a mesh is some additional meshes. Although it would be possible to build a three-dimensional mesh and thereby increase processing power and efficiency for 3-D spatial problems, such a system would still lack the capability to efficiently gather data globally from an image. An alternative is to let the additional meshes get progressively smaller, tapering to a point, thus forming a "pyramid." By connecting each PE to four "children" in the mesh below and a "parent" in the mesh above, a quadtree of interconnections is added to the mesh interconnections. The pyramid can then perform the computations of a tree machine if and when desired. For example, after some filtering operation has been applied to the image in the largest (bottom-level) mesh, the average value can be obtained by letting each PE compute the average value from its four children, until the global average emerges at the apex; the value is obtained in $O(\log N)$ time, whereas a pure mesh would require $O(N)$ time.

Pyramid machines also efficiently support multiresolution computations (Tanimoto, 1983; Rosenfeld, 1984; and Dyer, 1987) as well as hierarchical extensions to cellular logic (Tanimoto, 1984). These systems can also be thought of as specialized processors for manipulating pyramid data structures (see Article XIII.E1). Prototypes of pyramids have been constructed at the University of Washington (Tanimoto et al., 1987),

George Mason University (Schaefer et al., 1985), and are under development elsewhere (Cantoni et al., 1987). Closely related to the pyramid architecture is the mesh augmented by a tree without auxiliary meshes; an example of such a system is NON-VON, developed at Columbia University (Shaw, 1985).

Darpa Image Understanding Architecture. Another multilevel architecture based on a mesh is one developed at the University of Massachusetts (Levitan et al., 1987). This system was designed specifically for vision applications in which computation is to proceed in real-time at three levels of abstraction: the pixel (or low) level, the feature (or intermediate) level, and the symbolic (or high) level. The architecture calls for three corresponding processor levels: a mesh of 512×512 PEs, another mesh (64×64) of more powerful intermediate-level processors, and a collection of 64 LISP processors. Shared between the lowest two levels is a one-gigabyte dual-ported memory, whereas a 512-megabyte shared memory sits between the upper two levels.

The system is designed to efficiently support the algorithms developed for the VISIONS system (Riseman and Hanson, 1986), among others. A prototype is currently under development with the cooperation of Hughes Aerospace and sponsorship of the Defense Advanced Research Projects Agency.

The Connection Machine. Rather than augment a mesh with a tree or additional meshes, the Connection Machine uses a data-routing network, which is physically arranged as a hypercube. The general architecture of the system is given in Hillis (1985). The first version of the Connection Machine, the CM-1, became operational in 1986. That model allows either a 128×128 or a 256×256 array of processing elements to be installed. Each PE has 4K bits of local memory. The system operates from a 4MHz clock. The CM-2, available since the fall of 1987, uses 64K bits/PE and an 8MHz clock, plus optional floating-point hardware. The hypercube-based router of each model is 12-dimensional, with each router node responible for 16 PEs. However, the user programs data transfers as if each PE were accessible directly from any other. A good account of how the Connection Machine may be programmed for computer vision problems is given in Little et al. (1987).

Part of the inspiration for the Connection Machine was NETL (Fahlman, 1979), which is a model for a large hardware system based on a semantic-network/neural-network paradigm. Neural networks have also inspired research into a more amorphous family of information processing systems that are usually described as "connectionist."

Connectionist Architectures

The various models of computing that fall under the heading of *connectionist architectures* generally have their roots in observations of

human and other biological neural systems. In addition to the influence of neurophysiology and experimental psychology, the connectionist approach benefits from a recognition of some inherent limitations of conventional computers.

The "Von Neuman bottleneck" is the principal limitation of a conventional serial computer system. There is only one processing unit, the CPU, and it can perform only one operation at a time. These operations involve only one word-sized data object at a time, and memory can be accessed directly only by using addresses (not by contents, by semantic associations, or by structure). It is true that today's serial computers can perform an operation in 100 nanoseconds. Yet these operations are comparatively simple, and those required for artificial intelligence applications are complex enough to need thousands of the elementary operations. The result is that AI applications (and especially vision applications) run very slowly on Von Neumann–style computers.

Further underscoring the limitations of the traditional serial architecture is the fact that biological systems succeed at complex tasks even though their neural computing elements run several orders of magnitude more slowly than the corresponding electronic elements. The biological "proof" that parallelism works starts with the observation that a neuron requires on the order of one millisecond to fire, whereas computer switching times are on the order of 10^{-8} seconds (10 nanoseconds). To account for the computing power and intelligence of the human brain, we are forced to rule out the speed of the neuron as the key; the speed of human perception must be due to the brain's parallel architecture, not the speed of individual computing elements.

If we could have the same massive parallelism that we have in the brain, but with electronic computing elements instead of neurons, it seems that we should be able to obtain intelligent systems with 1,000 times the power of the brain. With systems of this power, what would take a human three years to learn might take such a computer only one day to learn, if the computer could somehow be provided with an efficient enough learning environment. The hope that man will be able to improve machine intelligence by building highly parallel, highly interconnected computer systems has stimulated considerable activity in connectionist research.

General Structure of a Connectionist System. A connectionist architecture consists of a specification for an elementary processing element, called a "unit" plus a specification of the interconnections among a collection of these units.

A unit may be thought of as a processor: a computing element that takes one or more inputs, maintains a state, and may produce one or more outputs. One of the inputs may be *external*, from outside the system; whereas other inputs to the unit may be the outputs or the states of other units, which are tied to the unit by *connections*. The set of states

that a unit can be in may be binary (i.e., the set {0, 1}), or it may be the set of real numbers or some interval of the reals or the integers, or it may be some other set. Many connectionist architectures use units that sum their inputs and then compare the sum with a threshold. Other systems use units that compute other, sometimes more complex, functions.

The connections among units are like the arcs of a graph; units are connected pairwise. Each connection from unit A to unit B is assigned a *weight* (or a *strength*). The weights are usually real numbers that regulate the influence that the state of one unit can have on the state of another. In some architectures, connections are constrained to be symmetric; in a symmetric-weight architecture, the connection from A to B always has the same weight as the connection from B to A.

An important aspect of some connectionist architectures is the manner in which the network changes over time. In addition to units changing state, the weights on the interconnecting arcs may change value.

Knowledge is represented in connectionist systems in different ways. In the "localist" approaches, each unit holds some knowledge. In the "wholistic" approaches, a given item of knowledge is represented as a configuration of several (and possibly all) units.

In the remainder of this section, we present several well-known types of connectionist networks and attempt to describe the manner in which they may solve problems.

The terms "connectionist architecture," "connectionist network," and "neural network" are often used interchangeably. We will often use the abbreviations "network," or "net" to refer to such a system.

Perceptrons. In the late 1950s and 1960s, a class of connectionist networks called *perceptrons* were studied (Rosenblatt, 1962). In the excitement of the day, great expectations were raised about the capabilities of perceptrons. Some negative results by Minsky and Papert (1969) triggered a backlash that subdued attention given to these systems for approximately a decade. Today there is a better understanding of perceptrons that makes it clear that many of the limitations cited by Minsky and Papert can be overcome by generalizing the model. (The introduction of "hidden units" into the networks is the key to increasing their power.)

Perceptrons have been most commonly studied as layered systems in which computations proceed bottom-up. Typically, input signals from sensors are fed up into the first layer, in which combinations of the inputs are weighted, summed, and thresholded to obtain a set of outputs from the first layer. These are subsequently weighted, summed, and thresholded in a second layer, etc., until the desired level of abstraction is reached. At that level, the inputs are classified (e.g., "Grandmother is in the picture").

Hopfield Nets. Whereas a layered perceptron typically produces each classification on a single separate output unit and therefore repre-

sents results *locally*, another approach is to represent results as global states of the network. This notion is combined with an iterative relaxation approach in the model of Hopfield (1982). In a Hopfield net, the units are started in a pattern of states that represents the input vector. Each unit then continually examines the units to which it is connected and computes a local energy function. Whenever this energy would be lowered by the unit's changing its state, it does so. Because the overall energy in the network decreases as long as there is activity, a Hopfield net must relax or converge. An analogous convergence criterion for relaxation labeling has been given by Hummel and Zucker (1983). The global state at which it converges represents the output.

Let us describe the Hopfield model more precisely. For a network of units connected symmetrically, the connection between unit i and unit j has a weight w_{ij}, which represents the extent to which the two units should attempt to be in the same state. A fixed threshold θ_i is associated with each unit. Let s_i denote the (current) state of unit i; that is, $s_i = 1$ if unit i is on, and 0 if it is off. Then the energy of the net (for a given state vector) is

$$E = -\sum_{i<j} s_i s_j w_{ij} + \sum_i s_i \theta_i$$

Each unit can compute the effect that its changing state would have on the total energy, using the formula,

$$\Delta E = E_{i\text{off}} - E_{i\text{on}} = \sum_j s_j w_{ij}$$

If the unit is off and ΔE is negative, it should turn on. If the unit is on and ΔE is positive, it should turn off; otherwise, it should maintain its current state.

To use a Hopfield net for pattern recognition, certain units can be designated as input units. After holding the input units in the input state until the rest of the system converges, the global state represents a local minimum configuration consistent with the input. This state may not be a global minimum.

Boltzmann Machines. To overcome the tendency of a pure Hopfield net with hidden units to become trapped in local minima that are not global minima, the transition of each unit from one state to another can be made probabilistic. By starting the relaxation at a high "temperature" in which transitions are almost completely random, and then gradually lowering the temperature so that transitions tend more and more to only reduce the system's energy, the probability of finding the global minimum can be made close to 1. This method, known as *simulated annealing*, was developed by Geman and Geman (1984) and independently with a different emphasis and name—Boltzmann machine—by Fahlman, Hinton, and Sejnowski (1983).

A Boltzmann machine is a computational system consisting of a set of elements called *units*. Each unit may be in either the 0 state or the 1 state, and it changes its state at each iteration (of a system cycle) stochastically according to the probability:

$$p_i = \frac{1}{(1 + e^{-\Delta E_i/T})}$$

where ΔE_i is the difference in energy between the 1 state and the 0 state of the ith unit, and T is a parameter analogous to temperature.

A Boltzmann machine can be thought of as a network of binary processors that use a form of the *Metropolis algorithm* (Metropolis et al., 1953) to update their states (Hinton and Sejnowski, 1987).

The Metropolis Algorithm. The Metropolis algorithm is a general procedure for finding the minimal energy state of a system by stochastically making local adjustments to it. It is a precursor of simulated annealing. The algorithm goes as follows:

Randomly select a state S.
Set $T \leftarrow$ initial temperature (high).
while $T > 0$ do
  Randomly generate an adjustment yielding state S'.
  Compute the energy difference: $\Delta E \leftarrow E(S') - E(S)$.
  If $\Delta E \leq 0$ then accept the state change: $S \leftarrow S'$.
    else accept it anyway with probability P:
    $P \leftarrow e^{-\Delta E/T}$.
    $x \leftarrow$ random number in [0,1].
    If $x < P$ then $S \leftarrow S'$.
  If there has been no significant decrease in E for many iterations then lower the temperature T.

An important element of such a procedure is the *temperature schedule*, which controls the gradual lowering of the temperature from one iteration to the next. Geman and Geman (1984) suggest the following schedule, where k is the iteration number and C is an appropriate energy constant:

$$T = C/\log(1 + k)$$

Clearly, in early iterations, when T is large, the system energy is permitted to increase often, thus allowing the system to escape from local minima. As T approaches zero, the system energy decreases almost monotonically; then the system "freezes" at a local minimum that is very likely to be the global minimum.

Application to Figure/Ground Discrimination. To illustrate how a stochastic-relaxation approach (which is based on a neural-network model) can solve problems in machine vision, an example is presented in which a figure/ground discrimination must be made. As

demonstrated in Kienker et al. (1986) and in Hinton and Sejnowski (1987), a parallel system can efficiently solve this problem even when the input information is noisy and incomplete.

A classical problem of visual perception is to take a binary (black and white) image and decide whether the black regions are figure and white regions background, or vice versa. The chalice of Rubin (Rubin's vase) is a particularly ambiguous case (see Figure C–2). The problem is just as difficult or more difficult when the black/white information is gone and only edge information is available.

Let us consider an array such as that shown in Figure D–2. Each square or triangle in the figure represents one unit. Each unit is connected to those immediately adjacent to it. The square units may be thought of as small regions, and the triangles represent oriented edges. A triangle that is on (white) corresponds to a strong edge, whereas one that is off indicates the lack of the corresponding edge. If a square is on, it is interpreted as belonging to the figure; otherwise, it is taken to be background.

The connections among units embody constraints about what constitutes a reasonable figure/ground interpretation. The weights are symmetric and isotropic (equivalent under 90-degree rotations). A square is connected to each of its eight nearest neighbors with weight +10. The

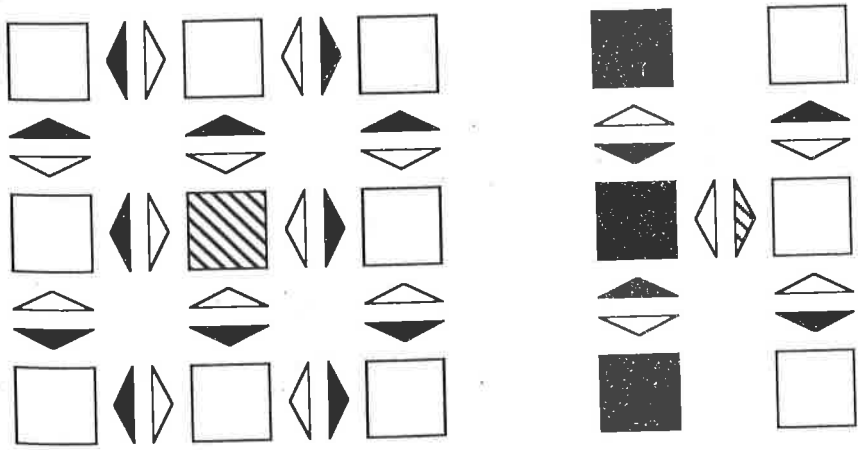

Figure D–2. Cell array for figure/ground resolution, showing (a) excitatory and inhibitory connections to a square (shaded), and (b) excitatory and inhibitory connections to a triangle (also shaded) Diagram after Sejnowski and Hinton (1987).

weight between a triangle and the square A it points to is $+12$, whereas for the one B that it points away from the value is -12. For each of the two squares on either side of A the weight with the triangle is $+10$, whereas for those on either side of B the weight is -10. The weight between an adjacent pair of triangles (facing in opposite directions) is strongly inhibitory (-15).

The input to the algorithm is an initial assignment of values to each unit. The inputs to the triangles represent the strengths of edges in an image, and they are called "bottom-up" inputs since they depend on the image data. On the other hand, the figure units are given initial weights "top-down" from an imaginary process that controls the focus of attention.

In the example shown, the edge elements bordering on a 9×6 rectangle were given initial inputs of 60; since those with values over 41 are shown in Figure D–3, this rectangle is visible. The top-down inputs to the figure units were given values according to a Gaussian distribution centered on the unit just to the right of the rectangle's center. The figure units shown are those with values exceeding 1.

Applying simulated annealing to this network, Kienker et al. (1986) found that it consistently converged on the desired solution. Figure D–4 shows their results. Although the method provides a useful demonstration of cooperative computation with simulated annealing, it breaks down on more complicated shapes such as spirals, unless a very long annealing schedule is adopted. However, figure/ground distinctions are also difficult for humans to make in cases of highly convoluted shapes like spirals.

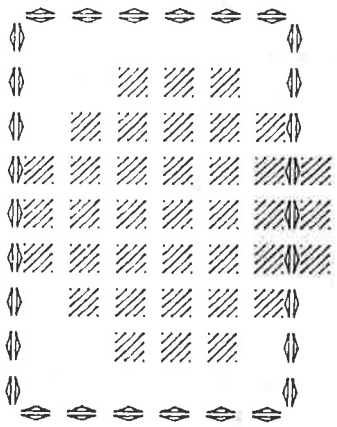

Figure D–3. Display of initial input values for the figure/ground problem. (Courtesy of T. Sejnowski.)

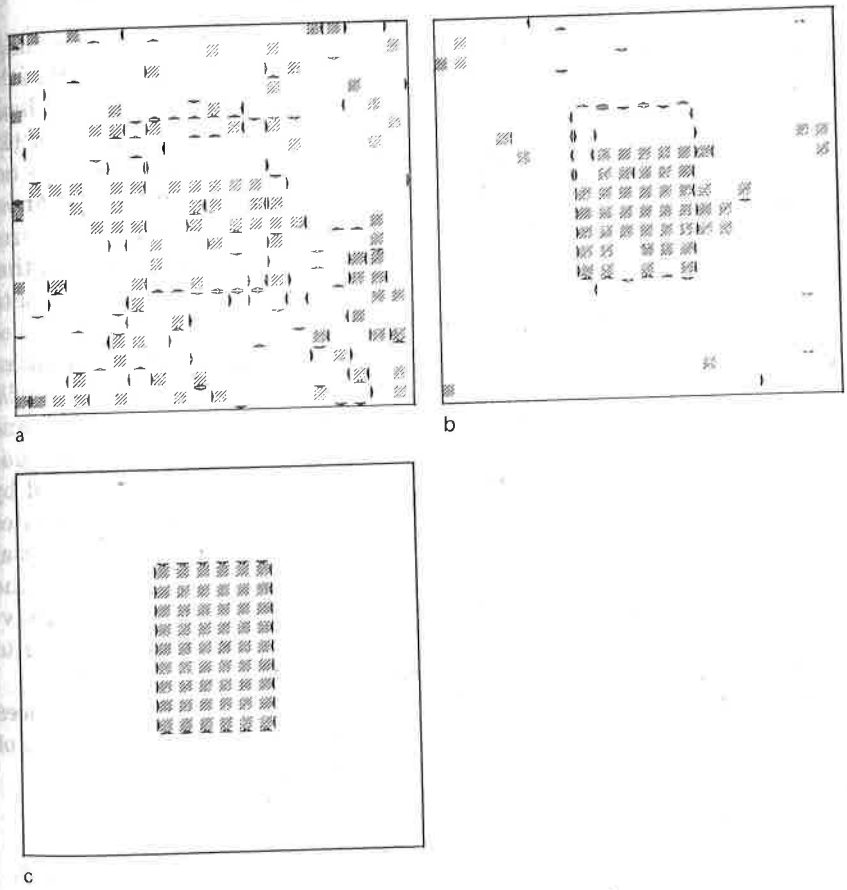

Figure D–4. Stages in the simulated annealing of the figure/ground problem: (*a*) after three iterations and $T = 16.2$, (*b*) after ten iterations and $T = 7.7$, and (*c*) after 28 iterations and $T = 3.3$. (Courtesy of T. Sejnowski.)

Multigrid Algorithms. Over the past decade it has been found that certain vision problems require the solution of two-dimensional numerical constraint satisfaction or optimization problems. Traditional numerical algorithms for these problems are computationally expensive. However, a class of numerical techniques called *multigrid* methods has been brought to the attention of the vision community by Terzopoulos (1984a). These methods make the solution of certain field reconstruction problems computationally much more attractive than they otherwise would be.

As noted in the preceding pages of this chapter, several vision prob-

lems boil down to computing a complete set of surface points or image pixels from sparse data. Stereo image analysis, for example, requires the determination of a depth map from a sparse set of depth values that have been determined by matching feature points in the two images (for example, Grimson, 1981, 1985). Since the sparse data is generally not sufficient to completely constrain the desired surface, assumptions about continuity of the surface are usually brought to bear on the desired solution. The resulting problem is one of finding the optimal surface that obeys the surface continuity constraints (which may allow for discontinuities) and the particular constraints imposed by the sparse data.

One formulation of this general reconstruction problem is as follows: imagine the surface to be reconstructed as the equilibrium state of a flexible plate that is supported by vertical pins of different lengths and attached to them by springs with different spring coefficients. The pins are irregularly spaced. The solution to the problem can be obtained by using a "variational principle" (Courant and Hilbert, 1953), which states that the equilibrium surface $u(x,y)$ is one that minimizes the potential energy of the system, which is composed of the energies due to the deformation of the plate, the springs, general external forces (e.g., gravity), external forces on the boundary, and bending moments applied to the boundary.

After approximation and discretization, the use of a finite-differences method to solve such a problem results in a large and sparse system of linear equations,

$$\mathbf{A}^h\mathbf{u}^h = \mathbf{f}^h$$

where $\mathbf{u}^h$ is the vector of nodal variables on the mesh using spacing h.

Although it is sometimes possible to solve such systems directly using Gaussian elimination or other methods to obtain an exact solution (up to machine precision), direct methods are more often than not inapplicable to realistic problems. For these cases, iterative techniques are required. Conventional iterative methods such as the Jacobi and Gauss-Seidel iterations continually update their current approximation, normally converging on the solution. Such convergence, however, is slow. On the other hand, multigrid methods perform their iterations at different levels of resolution in such a way as to accelerate the convergence.

The reason that the Jacobi and Gauss-Seidel methods converge slowly (when they converge) is that each local updating operation works on the neighborhood of a point in the mesh. Consequently excess energy or a deficiency of energy in the current approximation can move only one grid unit per iteration. This means that although high frequency components of the error surface can be damped rapidly, the low frequency portions require many iterations for their attenuation.

Multigrid relaxation achieves its acceleration of convergence by

allowing the low frequency components of the error surface to move rapidly across the space at coarse resolution levels. As in a pyramid data structure, a single neighborhood at a coarse level covers a large area in the finest level. Once a coarse level solution has been found, it can be projected into the next finer level as the starting approximation for a relaxation at that level.

Although the most obvious approach to multilevel relaxation (performing a sequence of conventional relaxation operations starting at a coarse level and progressing to the finest level) improves on unilevel relaxation, the best results are obtained by a more complex schedule of relaxation steps at different levels. Such schedules are discussed in Briggs (1987), and Hackbusch and Trottenberg (1982). One schedule is that implicit in the following two procedures adapted from Terzopoulos (1986):

```
procedure FullMultiGrid
  u^{h_s} ← SOLVE(s,u^{h_s},f^{h_s});
  for l ← s + 1 to L do
    v^{h_l} ← EXPAND(u^{h_{l-1}});
    MultiGrid(l,v^{h_l},f^{h_l});

procedure MultiGrid
  if l = s then u ← SOLVE(s,u, g)
    else
      for i ← 1 to n1 do
        RELAX(l,u,g);
        v ← REDUCE(u);
        d ← A^{h_{l-1}}v + REDUCE(g − A^{h_l}u);
        for i ← 1 to n2 do MultiGrid(l − 1,v,d);
        u ← u + EXPAND(v − REDUCE(u))
        for i ← 1 to n3 do u ← RELAX(l,u,g)
```

Here SOLVE applies unilevel relaxation long enough to achieve some desired degree of accuracy. RELAX applies a single unilevel iteration of the relaxation. The parameters n_1, n_2, and n_3 are set to obtain the best performance for a given class of problems. The coarsest level (or "starting" level) is indexed by s, and L is the index of the finest level. The vector $\mathbf{u}^{h_s}$ holds the approximation to the solution at the starting level. Vectors $\mathbf{u}$, and $\mathbf{v}$ hold current approximations at any level, with $\mathbf{v}$ one level coarser than $\mathbf{u}$ at any particular time. The matrices $\mathbf{A}^{h_l}$ and $\mathbf{A}^{h_{l-1}}$ represent versions of the original matrix $\mathbf{A}^h$ at resolution levels l and $l-1$, respectively.

The function EXPAND(u) takes a current approximation at level $l-1$ and produces an approximation at level l by using bilinear interpolation. Thus it maps data from one grid to the next finer grid.

Similarly, the function REDUCE(u) takes the approximation at level

l and produces a reduced-resolution version of it at level $l - 1$ using simple injection.

Multigrid methods have been applied by Terzopoulos to a variety of visual reconstruction problems including reconstruction of geometric surfaces, depth maps from stereo, lightness, and optical flow fields. The computational savings over unilevel relaxation were found to be quite significant; typically the time required for the multigrid approach was only two percent of that used by the non-multigrid method.

For addition information on multigrid algorithms see Terzopoulos (1986, 1984a), the tutorial by Briggs (1987), the collection of papers edited into a book by Hackbusch and Trottenberg (1982), and the seminal paper of Brandt (1977). For related work on relaxation in computer vision, see Article XIII.E4, and Glazer (1984).