The Chain Pyramid: Hierarchical Contour Processing

PETER MEER, MEMBER, IEEE, C. ALLEN SHER, AND AZRIEL ROSENFELD, FELLOW, IEEE

Abstract—**A novel hierarchical approach toward fast parallel processing of chain-codable contours is presented. The environment, called the chain pyramid, is similar to a regular nonoverlapping image pyramid structure. The artifacts of contour processing on pyramids are eliminated by a probabilistic allocation algorithm. Building of the chain pyramid is modular, and for different applications new algorithms can be incorporated. We describe two applications: smoothing of multiscale curves, and gap bridging in fragmented data. The latter is also employed for the treatment of branch points in the input contours. A preprocessing module allowing the application of the chain pyramid to raw edge data is also described. The chain pyramid makes possible fast, O[log (image_size)], computation of contour representations in discrete scale-space.**

Index Terms—**Chain-codes, contour smoothing, image pyramids, parallel processing.**

I. Introduction

A CONTOUR in a binary image is a string of (say) black pixels, each having at most two black neighbors. Such a string is a chain-codable curve [12]. The problem of branch points on curves, i.e., pixels with more than two neighbors, will be discussed later in the paper. In a two-dimensional image, contours convey information about the shapes of objects present in the image. Fast extraction and analysis of contours from a digital image requires that the processing take place in a parallel hierarchical environment. Different specialized parallel architectures for contour extraction have been proposed [1], [32]. Hartmann [14] described a hierarchical contour coding scheme in the context of pattern recognition in the visual system.

The most frequently employed parallel hierarchical processing environment is the *image pyramid*. In a pyramid, features of a region of the input image can be extracted in O[log (region_size)] processing steps. We propose a new method for extracting and analyzing contours in digital images based on a new type of image pyramid, the *chain pyramid*. In the chain pyramid the contours are represented as linked lists. This allows flexibility in performing analysis and/or processing of the contours. Processing can be simultaneous with the extraction procedure.

The basic operation in contour extraction is *tracing*. A pixel in the input image becomes connected to (at most two) neighbors by verifying connectivity constraints in a 3×3 neighborhood. The concatenation of these local operations leads to the delineation, or tracing, of the contour. Ullman [33] proposed tracing as one of the fundamental visual routines employed for the description of spatial relationships in the two-dimensional world. Contour tracing, or its more general counterpart, connected component labeling, can be performed in logarithmic time in pyramidal environments [22].

Let a binary input image have $2^n \times 2^n$ pixels with the black pixels constituting the contours to be extracted. The simplest pyramid structure suffices for contour extraction. At every level of the pyramid the cells are organized in an 8-connected mesh. (For contour extraction 8-connectedness is preferred to the simpler 4-connected case.) A cell, or *parent*, at level l of the image pyramid, is also connected with a field of 2×2 cells, its *children*, at level $l - 1$. Each of these children in turn is connected with 2×2 cells at level $l - 2$, etc. The hierarchy of connections defines the structure of the pyramid.

Successive levels of the pyramid contain fewer pixels, so that they can only be a reduced resolution representation of the input because the capacity of the cells' memory is limited. The first level has $2^{n-1} \times 2^{n-1}$ pixels, the second level $2^{n-2} \times 2^{n-2}$ pixels, and so on. After $n =$ log (image_size) levels the apex of the pyramid (containing only one cell) is reached. To achieve O[log (image_size)] processing time, the processing should be local and recursive. A parent at level $l + 1$ must only employ information available to its children at level l and it neighbors on the mesh at level $l + 1$. The local operations performed by a parent on the information provided by its children define the nature of the reduced resolution representations derived from the input image.

Ullman's approach toward contour tracing as a visual routine led to work in which connectivity is established by *coloring* of pyramid cells containing only one contour fragment. In the algorithm proposed by Edelman [9], the parents compute a topological characteristic, the Euler number, of the represented image region. If the region is found to contain only one connected component the procedure is repeated at the next level, i.e., the same color is allocated, if possible, to a larger region. Mahoney [19] implemented an algorithm based on fusing the 2×2 field of children into two subregions. The subregions are chosen depending on the local configuration of the contour. The final result of both coloring based methods is a description of the curve by pyramid cells at different levels. This representation is very sensitive to the shape of the

Manuscript received September 1, 1988; revised July 7, 1989. This work was supported by the National Science Foundation and the U.S. Air Force of Scientific Research under NSF Grant DCR-86-03723.

The authors are with the Center for Automation Research, University of Maryland, College Park, MD 20742.

IEEE Log Number 8932053.

contour, and to the position of the curve relative to the boundaries of higher level cells in the pyramid. Both Edelman and Mahoney discuss the problem of position dependence for some of their results.

Kropatsch [18] represented contours at multiple resolution by introducing a grammar of the intersections between a contour fragment and the four boundaries of a pyramid cell. He employed two complementary pyramids, the second one having its cells rotated by 45 degrees. Adjacent cells taken from either pyramid are then merged together based on a large set of rules. The method is cumbersome, yielding high sensitivity of the representation to the structure of the pyramids.

In our method we approach the problem differently. We reduce as much as possible the influence of the pyramid structure on the efficiency of the tracing procedure by representing the curve fragments as lists, rather than through the topology seen within a cell's boundaries. As a result, the contour tracing can easily be combined with other parallel algorithms for curve processing. We discuss two such algorithms: smoothing of multiresolution contours, and bridging gaps in fragmented contours.

In Section II we describe several problems confronted by any algorithm for contour tracing in a hierarchical environment, and present the solutions employed to avoid them. In Section III the different modules coupled with contour tracing are presented together with experimental data. In Section IV a preprocessing module, taking the thresholded output of an edge detector and generating input suitable for the chain pyramid, is described. A short discussion of further directions of development is given in Section V.

II. Contour Tracing in a Hierarchical Environment

The most important property of an image pyramid is the gradual transition from global to local processes. Large regions in the input are represented by many cells at lower levels of the pyramid. The same region, however, is seen by only one cell near the apex of the pyramid. Thus, at higher levels cells must trace larger and larger contour fragments. The capacity of a cell, however, is limited and at higher levels a complete description of the contour fragment within the receptive field is usually not possible. To avoid overloading of the cell's processing capacity, contour tracing in hierarchical environments must be combined with a data compression procedure.

When a cell sees only one long contour fragment, data compression is immediate by extracting the properties of interest from the part within the cell's boundaries. Those properties should be computable recursively from the information available to the children.

More serious cases of overload are shown in Fig. 1. The square grid drawn with the thinnest lines illustrates the structure of the pyramid base. The receptive fields of cells from different pyramid levels (i.e., the region on the base of the pyramid integrated by the cell) are shown by lines of different thickness. A thicker line is the boundary for a cell at a higher level. The two thickest lines, intersecting at the center, represent the common boundaries of the four cells one level below the apex. We will call these lines the midlines of the pyramid. To facilitate the recognition of the receptive field boundaries, the pixels belonging to the contour are only hashed and not made black.

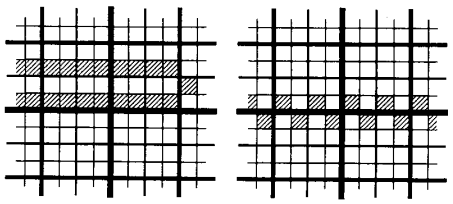


Fig. 1. Examples of contours overloading a cell's capacity at higher pyramid levels. Thicker lines represent boundaries of higher level receptive fields. The pixels belonging to the contour are hashed. (a) Multiple fragments in the receptive field. (b) The worst case.

In Fig. 1(a) the cells at the first pyramid level (2×2 receptive fields) see one contour fragment. The cells at the second level (4×4 receptive fields at the base) may see two separate contours and cannot assess whether the fragments belong to the same curve or not. In the coloring based algorithms this type of overload is avoided by stopping the coloring and creating a root once multiple fragments are detected. As a result, the contour is represented by roots at different levels of the pyramid and further processing of the curve may become cumbersome since information is distributed across the whole structure.

In the chain pyramid, the contour tracing is driven by the contour itself and not by its spatial relationship with the underlying pyramid structure. Two contour fragments within the same receptive field are treated as separate entities and their (compressed) descriptions are carried forward in the pyramid. Higher levels of the pyramid then connect the two fragments if necessary. The limit on the number of fragments processed by a pyramid cell is set only by the processing capacity of the cell. If the number is sufficiently large, the total configuration becomes a texture. Textures can be represented by statistical characteristics which achieve a much higher compression ratio.

The type of overload illustrated by the contour in Fig. 1(b) is more severe. Let a contour be of length $N = 2^n$, i.e., the size of the image side. The contour is located at the center of the image and crosses the principal boundaries of the pyramid $N - 1$ times. If local contour tracing is kept within the boundaries of the pyramid cells, no data compression is possible for this contour up to the apex of the pyramid. The apex then sees N separate curve fragments and must perform O[image_size] operations to connect them. Thus, without additional precautions, for the contour in Fig. 1(b) the image pyramid has the cells at the higher levels overloaded, and cannot accomplish the processing in O[log (image_size)] time, violating the fundamental property of such structures.

Because of the distributed processing architecture of the image pyramid, proposed methods of avoiding overload in connected component applications [8] cannot be applied. We now present a simple parallel probabilistic algorithm based on local ordering of random variables, which assures logarithmic processing time even for the worst case shown in Fig. 1(b).

A. Probabilistic Allocation Algorithm

All the black pixels defining the contour in Fig. 1(b) are equivalent. They are *siblingless*, i.e., there is only one child per parent belonging to the contour. The following important *connectivity property* of siblingless child-cells always holds:

> In a string of siblingless child-cells, every other cell can be removed without breaking connectivity in the reduced resolution representation at the parents' pyramid level.

To prove the connectivity property it suffices to remark that for a siblingless child the parents of its two neighbors in the string are adjacent on the next level. Thus, if the siblingless child is removed, the string at the next level remains contiguous. In Fig. 2 more examples of siblingless child-cells can be found at level 0, and the above property can be verified. The overload of higher pyramid cells for the contour in Fig. 1(b) is avoided if the information about every second contour pixel is allocated in parallel to their neighbors along the string. No propagation, however, with a cell simultaneously receiving information from one of its neighbors and allocating information to the other, should occur.

Assume now that we have labeled the string of siblingless pixels (cells) in Fig. 1(b) with a binary sequence of the form

$$\cdots\ 010101010101\ \cdots \tag{1}$$

All the cells which received a 1 label can then become the receivers of information from the cells having 0 labels. The connectivity property is satisfied because only every other cell is removed from the string, and the contour is represented at the next, lower resolution contiguously. Repeated application of the allocation procedure at subsequent levels will eliminate overloading of the processing capacity of higher pyramid level's cells.

Unfortunately such a binary sequence cannot be derived from the absolute addresses of the pyramid cells because the structure of the pyramid is independent of the shape of the contour. Nevertheless, a sequence similar to (1) can be generated in parallel in at most three iterations. The method is based on local ordering of independent random variables. Because of the involvement of random processes, we call it the *probabilistic allocation algorithm*. Only a short description of the algorithm is given here; for more detail see Meer [21].

The algorithm assumes that local connectivity has already been established among the pixels belonging to the contour. (How this is done will be described in the next section.) The contour can now be regarded as a one-dimensional string with each cell having access to its two neighbors. (The existence of end-points with only one neighbor is not relevant for the moment.) Let a cell j have access to three variables: p_j, q_j, and x_j. The variables p_j and q_j are one bit binary numbers. The label of the cell is defined by the value of p_j. The variable x_j is an outcome of the random variable $\boldsymbol{x}_j$ which has a continuous uniform probability density between 0 and 1. Random variables allocated to different cells are independent and identically distributed (i.i.d.).

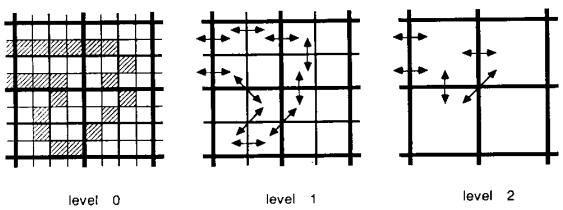


Fig. 2. Example of embedding the doubly linked lists carrying representations of a contour on different chain pyramid levels. The arrows represent the pointers of the list.

Initially all labels $p_i = 0$, and each cell draws an outcome of its random variable. The label is changed from 0 to 1 whenever the outcome x_j is the largest in the neighborhood of the cell:

$$\begin{aligned} p_j &= 1 \quad \text{if } x_{j-1} < x_j > x_{j+1} \\ p_j &= 0 \quad \text{otherwise.} \end{aligned} \tag{2}$$

Note that the probabilistic rule (2) excludes generation of two adjacent 1's. The probability that a cell has its label changed to 1 is $1/3$. This probability is too low to generate a sequence similar to (1), and iterative application of the rule is necessary.

Let the number of the current iteration be k. The second variable $q_j(k)$ describes the concatenated state of the local configuration of cells $j-1, j, j+1$. The variable is set to 1 whenever all three cells have 0 labels:

$$\begin{aligned} q_j(k) &= 1 \quad \text{if } p_{j-1}(k-1) = p_j(k-1) \\ &\qquad\qquad = p_{j+1}(k-1) = 0 \\ q_j(k) &= 0 \quad \text{otherwise.} \end{aligned} \tag{3}$$

At the beginning of the kth iteration new outcomes $x_j(k)$ are drawn and the value of $q_j(k)$ is determined. The updating rule (2) is then applied only to the cells with $q_j(k) = 1$. It can be shown [21] that after $k = 2$ iterations the length of the longest run of consecutive 0's is 3, with a residual probability around 10^{-5}. These runs can be eliminated by deterministically changing 0's to 1's if their two neighbors are both 0's.

The sequence of labels allocated to the string of contour pixels thus has only two local configurations: 101 and 1001. For the 101 configuration, the content of the cell labeled 0 is split between the two receiving neighbors.

TABLE I
NUMBER OF SEPARATE CONTOUR FRAGMENTS TO BE PROCESSED BY THE APEX

Without Probabilistic Allocation				
Length of the contour	32	64	128	256
With Probabilistic Allocation				
Average	2.55	2.69	2.52	2.46
Standard deviation	0.589	0.628	0.608	0.537
Maximum	4	4	4	4

For the 1001 configuration the cell with 0 label but larger outcome x_j becomes a receiving cell and has its label changed to 1. The content of the other cell with label 0 is then split between its two receiving neighbors. Note that in both cases connectivity in the next level's representation is preserved because only every other child belonging to the contour is eliminated. The number of siblingless cells can only decrease at higher levels of the pyramid. The probabilistic allocation algorithm is repeated at every level after local connectivity is established and most of the siblingless cells which survive the algorithm at the first level are eliminated at subsequent levels.

To measure the effectiveness of the probabilistic allocation algorithm, the worst case contour [Fig. 1(b)] was processed in the chain pyramid for contour lengths of 32, 64, 128, and 256 pixels. The number of separate contour fragments arriving to the apex of the pyramid was taken as quantitative measure. Recall that without probabilistic allocation, this number is equal to the length of the curve. For each case 100 trials were run. The experimental results are shown in Table I.

From Fig. 1(b), it can be seen that a perfect allocation would always yield two contour fragments to be processed by the apex. The length of the contour should not be a relevant parameter. The experimental results confirm this. The average number of contour fragments is always close to 2.5 with a standard deviation of 0.6. The largest number of contour fragments never exceeds four, corresponding to one fragment in each of the four quadrants.

Note that the probabilistic allocation algorithm can be applied successfully to compensate for the other artifacts created by the rigid pyramid structure (like erroneous blob detection), independently of the local property employed in the definition of siblinglessness.

III. CONTOUR PROCESSING ALGORITHMS

In this section we describe several algorithms employed for processing of contours in the chain pyramid. An important algorithm, probabilistic allocation, was described above. The algorithm preceding the allocation procedure, local connectivity determination, is discussed below. Data compression algorithms must always be employed to build the pyramid, but these algorithms depend on the specific application. We present a smoothing algorithm for multiscale contours. A different application, gap bridging of fragmented contours, is then discussed to illustrate the modularity of processing in the chain pyramid environment. The gap bridging algorithm is also useful for the treatment of branch points, i.e., pixels having more than two neighbors.

A. Local Connectivity Algorithm

The hierarchical environment built to extract and represent contours is called a *chain pyramid* because the fundamental procedure, the local connectivity algorithm, makes use of the chain-codable property of the contours.

At the base of the pyramid the black pixels defining the contour have at most two neighbors of the same color. A pixel establishes connection with its neighbor(s) by defining two pointers. The pointers are based on the relative position of the two pixels and do not make use of the pixels' absolute addresses in the mesh at level 0. For endpoints the second pointer points back to the pixel from which it originates. The contour pixels generate the pairs of pointers in parallel and the string becomes locally connected. The resulting data structure is known as a *doubly linked list* [16, p. 140]. In a doubly linked list a node is aware of its two neighbors, but cannot discriminate their positions in the list relative to a *global* scanning direction. The doubly linked list derived from the contour is distributed across the cells of level 0, the base of the pyramid. Every cell with a black pixel carries one node of the list and the two pointers. The doubly linked list constitutes the input to the chain pyramid.

In Fig. 2 the pixel string of a contour at the base (level 0) of the pyramid is shown. The same convention as in Fig. 1 is used to illustrate the structure of the pyramid. The parents at the first level (2×2 receptive fields) verify whether they see only one child belonging to the contour. These siblingless children are labeled 0, while every other pixel receives a 1 label. The probabilistic allocation algorithm then eliminates most of the siblingless children by replacing strings of $\cdots$ 00000 $\cdots$ with $\cdots$ 01010 $\cdots$ strings.

After the allocation procedure is accomplished, the parents at level 1 sequentially scan their receptive fields for contour fragments. Whenever a black pixel is detected it must belong to the distributed doubly linked list created at level 0. The contour fragment is then traced to the boundaries of the field, and the parent creates a pointer to the neighbor at level 1 whose child was reached by the tracing. The tracing operation makes use of the local connections recorded in the doubly linked list, and involves only a few steps along the list. As a result, a new doubly linked list, distributed across the cells of level 1, is created. The nodes of the list carry the representation of the contour at the reduced resolution of level 1. A data compression algorithm reduces the amount of information retained by every cell (list node) for processing at subsequent pyramid levels. In Fig. 2 a possible embedding of the doubly linked list of the contour at pyramid level 1 is shown. Note the efficiency of the probabilistic allocation algorithm in eliminating siblingless children.

The building of the chain pyramid is a recursive process. To obtain level 2 the steps described above are ap-

plied to level 1. A cell at level 1, however, may carry several curve fragments, each being represented by a part of the doubly linked list. The parents verify the number of relevant children, i.e., children carrying representations of at least one curve fragment. If siblingless children are detected, the probabilistic allocation algorithm reduces their number, making use of the connectivity recorded in the doubly linked list distributed across level 1. The parents then scan for contour fragments carried by each of the four children. If a yet unprocessed curve fragment is reached, it is traced through the receptive field by following the pointers of the doubly linked list of level 1. Whenever the tracing leaves the receptive field, a new pointer to the corresponding neighbor is created at level 2. The parents handle multiple curve fragments within their receptive fields as separate entities unless they can connect them. A possible embedding of the level 2 doubly linked list of the contour is shown in Fig. 2.

The local connectivity assessment algorithm is applied recursively until the doubly linked list is reduced to one *root* cell. Note that the root cell does not coincide with the apex if the contour does not cross a midline of the pyramid.

B. Data Compression and Smoothing Algorithms

The simplest data compression is achieved when global characteristics of the contour, such as length and average curvature, are computed. Any recursively computable global characteristic can be employed. The doubly linked list representations control the way children's information is combined along the contour. The global characteristic is obtained in the root cell after $O[\log(\text{image\_size})]$ processing time.

A more challenging data compression task is *contour smoothing*. A contour at the base of the pyramid is a string of black pixels, each with an integer address on the discrete grid of the underlying square lattice. The quantization introduces artifacts, and chain-codable contours have a ragged appearance (see for example the contour at level 0 in Fig. 3). To achieve good perceptual qualities the contours must be smoothed.

Multiscale curves are another important class of contours which require smoothing. These curves convey information at several levels of resolution. At level 0 in Fig. 4, a multiscale curve is shown. The curve is perceived at low resolution as a band with two 90-degree turns, while at high resolution the wiggling pattern becomes evident. To extract the low resolution trend of the band, the curves must be smoothed.

The smoothing algorithm has two distinct parts. The first part is the mandatory data compression; the parents retain only the essential information about their children. The amount of smoothing achieved by data compression, however, does not suffice, and in the second part of the algorithm additional smoothing is performed along the contour, on the mesh at the parent's level.

A pixel at level 0 (the base) has access to its address on the mesh. Recall that for building the chain pyramid the absolute addresses are not necessary. Each parent at level 1 computes the line centroid of the contour fragment within the boundaries of its receptive field. The centroid

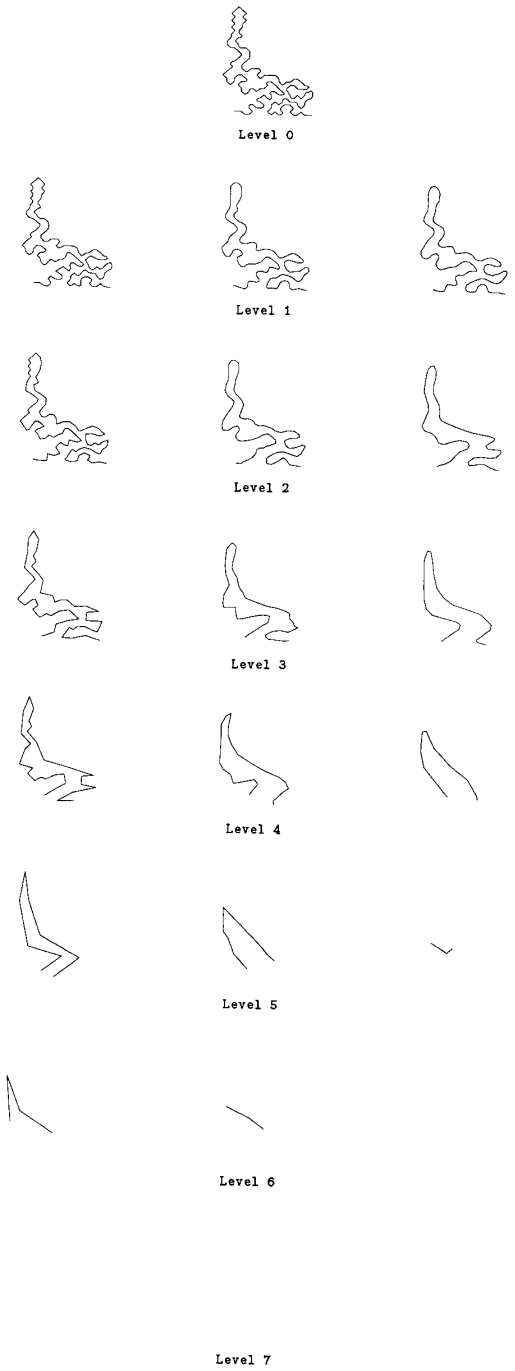


Fig. 3. Smoothing of a chain-codable contour. *Left column:* Only data compression is employed to build the next pyramid level. *Middle column:* Both data compression and smoothing along the contour are employed. *Right column:* All the three algorithms, data compression, smoothing along the contour, and probabilistic allocation are employed. Once the contour reaches its root cell it is no longer drawn.

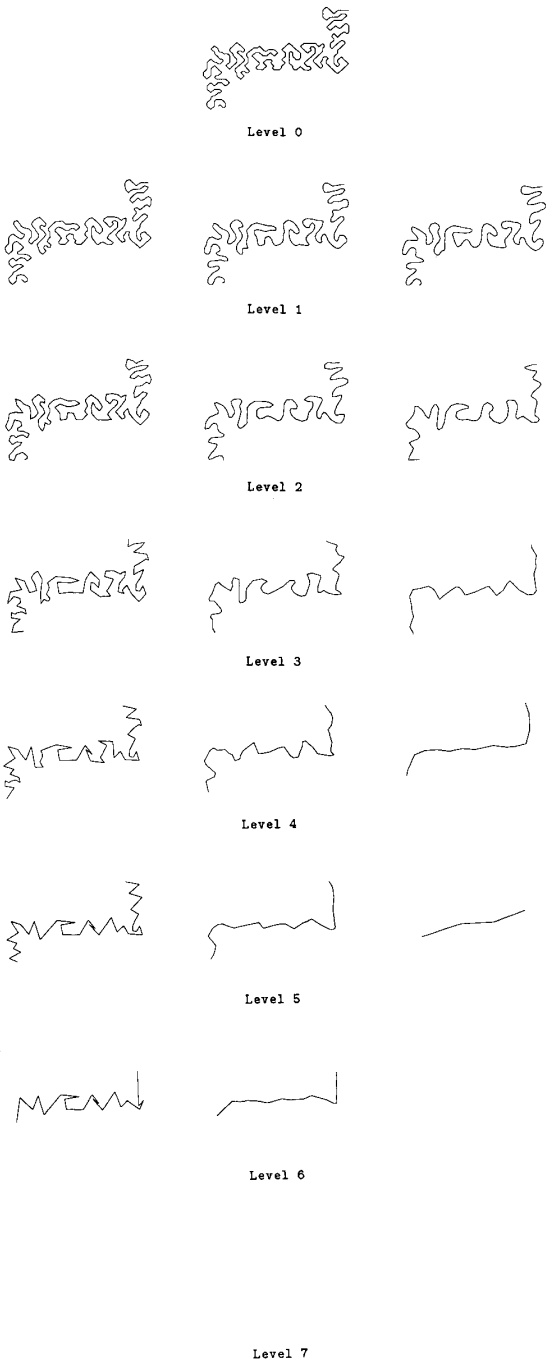


Fig. 4. Smoothing of a multiscale curve. Same as Fig. 3.

has the coordinates

$$x(i_1, j_1) = \frac{\sum_{R_1} x(i_0, j_0)}{|R_1|} \qquad y(i_1, j_1) = \frac{\sum_{R_1} y(i_0, j_0)}{|R_1|} \tag{4}$$

where R_1 is the ensemble of $|R_1|$ contour (black) pixels within the 2×2 receptive field of the parent; indexes with subscript 0 and 1 refer to level 0 and 1, respectively. The centroid's coordinates are no longer integer numbers corresponding to sites of the lattice at level 0, and the centroid does not necessarily lie on the contour. However, it is always inside the convex hull of the contour fragment [25, p. 100]. Thus smoothing is achieved if the contour is replaced by the polygonal line obtained by connecting the centroids. The polygonal line can be drawn directly from the doubly linked list of level 1, where the nodes carry the centroid coordinates and the structure of the contour is reproduced by the pointers. No new curvature extrema can be created by this procedure.

For higher levels the formulas (4) must be generalized. Each cell at level l stores the centroid coordinates $x_{kc}(i_l, j_l)$ for each one of the kc contour fragments within its $2^l \times 2^l$ receptive field. The length (at the base) of each contour fragment, $n_{kc}(i_l, j_l)$, computed as the number of pixels in its string, is also available. The parents at level $l + 1$ compute the new line centroid coordinates $x_{kp}(i_{l+1}, j_{l+1})$ representing the kp contour fragments in the new $2^{l+1} \times 2^{l+1}$ receptive field:

$$x_{kp}(i_{l+1}, j_{l+1}) = \frac{\sum_{R_{kp,l+1}} n_{kc}(i_l, j_l) x_{kc}(i_l, j_l)}{\sum_{R_{kp,l+1}} n_{kc}(i_l, j_l)}$$

$$y_{kp}(i_{l+1}, j_{l+1}) = \frac{\sum_{R_{kp,l+1}} n_{kc}(i_l, j_l) y_{kc}(i_l, j_l)}{\sum_{R_{kp,l+1}} n_{kc}(i_l, j_l)} \tag{5}$$

where $R_{kp,l+1}$ is the ensemble of contour fragments in the receptive field of the parent, which can be connected when tracing the kpth contour part. The updated contour fragment lengths are also computed:

$$n_{kp}(i_{l+1}, j_{l+1}) = \sum_{R_{kp,l+1}} n_{kc}(i_l, j_l). \tag{6}$$

All the information is stored at the node of the doubly linked list carried by the parent. Connectivity in the reduced resolution representation of level $l + 1$ is accurately reproduced by the pointers of the parents' list. The smoothed contour corresponding to level $l + 1$ of the chain pyramid is then obtained by drawing line segments between the centroids stored at consecutive list nodes.

At level 0 in Figs. 3 and 4, two different chain-codable contours defined in 128×128 images are shown. In the left column of Figs. 3 and 4 the smoothed contours obtained from the different chain pyramid levels are given. To illustrate only the effect of data compression, the probabilistic allocation algorithm was not used in building the chain pyramid. As can be seen, the contours at higher levels remain ragged because of the siblingless children situated close to the midline of the pyramid. Note especially the contours obtained for levels 4, 5, and 6 in Fig. 4.

The amount of smoothing is improved if after data compression the cells at level $l + 1$ exchange information on the mesh at that level. For each contour fragment stored

in a cell the two neighbors along that fragment are accessed through the pointers of the doubly linked list. The ensemble of three cells is denoted by $Q_{kp,l+1}$. New line centroid coordinates, $(X_{kp}(i_{l+1}, j_{l+1}), Y_{kp}(i_{l+1}, j_{l+1}))$, are then computed based on the information available to the cell and its two neighbors:

$$X_{kp}(i_{l+1}, j_{l+1}) = \frac{\sum_{Q_{kp,l+1}} n_{kp}(i_{l+1}, j_{l+1}) x_{kp}(i_{l+1}, j_{l+1})}{\sum_{Q_{kp,l+1}} n_{kp}(i_{l+1}, j_{l+1})}$$

$$Y_{kp}(i_{l+1}, j_{l+1}) = \frac{\sum_{Q_{kp,l+1}} n_{kp}(i_{l+1}, j_{l+1}) y_{kp}(i_{l+1}, j_{l+1})}{\sum_{Q_{kp,l+1}} n_{kp}(i_{l+1}, j_{l+1})}. \quad (7)$$

For the computation of the next level's centroid coordinates (5) the new values are employed. Note that the lengths of the curve fragments were not updated along the mesh. The effect of the additional smoothing, without the probabilistic allocation algorithm being applied for pyramid building, is shown in the middle column of Figs. 3 and 4. The amount of smoothing increases, but artifacts due to siblingless children are still present. The right column in Figs. 3 and 4 shows the smoothed contours obtained in the chain pyramid when the probabilistic allocation algorithm is also present in the pyramid building process. The results are much improved, and for the multiscale curve the trend (Fig. 4, level 4) is correctly extracted.

As an artifact of the smoothing we should mention the shift of the endpoints toward the center of the contour at higher chain pyramid levels. This artifact is always present when a convolution window must be entirely contained inside the data sequence. The root cell of the contour is reached whenever both endpoints are in the same receptive field. In Figs. 3 and 4, once the contour is represented by its root cell, no low resolution representation is drawn. The more efficient the smoothing, the lower the root cell is located in the pyramid.

For efficient smoothing of a planar curve, the curve must be described in terms of its arc length from one of the endpoints. An averaging window is then moved in one dimension (that of the arc length) along the curve. This is the case in several scale-space applications in which smoothing is achieved by repeated convolutions with Gaussian windows of increasing spread [2], [24], [27]. The convolution is implemented as a sequential scan along the contour, and the results are kept at all the locations.

The corrections introduced by the probabilistic allocation algorithm yield the best possible representation of the one-dimensional structure of a contour in a parallel two-dimensional processing environment. Thus, the chain pyramid can be employed to build scale-space representations of contours in O[log(image_size)] instead of O[image_size] time, by performing the convolutions in parallel. Gaussian shaped windows can be obtained with Burt's [3] fast filtering method. To have full resolution scale-space representations, the capacity of the pyramid cell's memory must be increased. This increase does not affect the speed of the processing, if local subhierarchies are defined [20].

C. Gap Bridging Algorithm

We now extend the chain pyramid to allow processing of contours with branch points, that is, with pixels having more than two black neighbors. An example of such a contour is shown in Fig. 7(a). The branch points appear at the intersections of contour fragments. Eliminating the pixels with more than two neighbors creates gaps, and yields a segmentation of the image.

The removal of the branch points takes place at the base, level 0, of the pyramid. In many cases the segmentation into branches is not of interest, and in the lower resolution representations the correct continuity of the contour fragments should be restored. Recall that the representation of each contour fragment is a doubly linked list distributed across the pyramid cells. The continuity is restored by creating new pointers between the two lists corresponding to the two separate contour fragments in the image. This is done by the *gap bridging algorithm*. The number of removed branch points should be minimal, and the chain-codable contour fragments should be kept as close as possible, to have the gap bridging efficient.

In Fig. 5(a) an example of a complex branch point is shown. The *branch point removal procedure* is part of the local connectivity algorithm at level 0. Recall that the algorithm begins by finding black neighbor pixels, and a count of the number of these pixels is an integral part of the procedure. All the pixels with more than two neighbors are then removed in parallel. (In Section IV a slight refinement, required for contours extracted from edge data, will be described.) The removal of such pixels from Fig. 5(a) yields the configuration in Fig. 5(b). The contour is broken into eight separate fragments, denoted by different hashings. The endpoint of each contour fragment, however, can extend the fragment to its removed neighbor without violating the chain-codability condition. The extensions are achieved in parallel by creating the corresponding pointers in the doubly linked lists. The ensemble of extended contour fragments, shown in Fig. 5(c), has the minimum number of pixels removed from the original image. The result of the branch point removal procedure applied to the contour in Fig. 7(a) is given in Fig. 7(b). This image is the input to the gap bridging algorithm of the chain pyramid.

The coarse quantization of the contour fragments in the input image prohibits gap bridging at the highest resolution. The contour fragments must be smoothed first before local continuity across a gap can be assessed. Multiresolution interpolation techniques are available in the literature [4], [15]. In the chain pyramid, we employ a different method for gap bridging which better fits the available contour representations. The modularity of processing in the chain pyramid allows the gap bridging algorithm to be

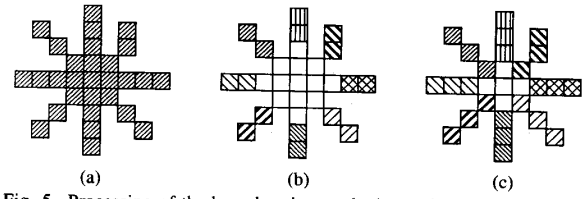


Fig. 5. Processing of the branch points at the base of the chain pyramid. (a) Example of a branching contour. (b) Removal of all the pixels with more than two neighbors. The different contour fragments are shown by different hashing textures. (c) Extension of the contour fragments.

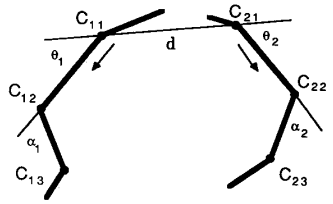


Fig. 6. Definition of the parameters employed in the gap bridging algorithm. The thin lines are drawn to show the definition of the angles. The arrows correspond to the scanning direction along the contour fragments.

intercalated between the data compression and smoothing procedures described in the previous section. As we have shown, smoothing of the contour is achieved by recursively computing the line centroid coordinates for contour fragments of increasing lengths. The gap bridging algorithm makes use, on a contour fragment, of the three centroids nearest to the gap.

The parameters of the algorithm are defined in Fig. 6. Let C_{11}, C_{12}, and C_{13} be the three centroids nearest to a gap on a contour fragment. Similarly, C_{21}, C_{22}, and C_{23} are the three nearest centroids on the other fragment. The polygonal lines of the contour fragments are considered as oriented away from the gap (see the arrows in Fig. 6). This scanning direction is always available because we are at the end of the doubly linked list representing the contour fragment. Four angles, with positive values corresponding to counterclockwise rotation, can be defined by the oriented directions

$$\theta_1 = \overline{C_{21}C_{11}C_{12}} \qquad \theta_2 = \overline{C_{11}C_{21}C_{22}}$$
$$\alpha_1 = \overline{C_{11}C_{12}C_{13}} \qquad \alpha_2 = \overline{C_{21}C_{22}C_{23}} \tag{8}$$

where the order of the centroids defines the sign of the angle.

Assume that data compression has already been performed at the parents' level and the coordinates of the centroids were computed by (5). A pair of contour fragments is considered for gap bridging if the following conditions involving the angles in (8) are *simultaneously* satisfied:

$$\Delta_1 = |\theta_1| + |\theta_2| < \frac{\pi}{4} \tag{9}$$

$$\Delta_2 = \big||\alpha_1| - |\theta_1|\big| < \frac{\pi}{4} \tag{10}$$

$$\Delta_3 = \big||\theta_1| - |\theta_2|\big| < \frac{\pi}{4} \tag{11}$$

$$\Delta_4 = \big||\theta_2| - |\alpha_2|\big| < \frac{\pi}{4} \tag{12}$$

$$\Delta_5 = \big||\alpha_1| - |\alpha_2|\big| < \frac{\pi}{4}. \tag{13}$$

A parent may see several contour fragments within its receptive field, and conditions (9)–(13) must eliminate most of the possible combinations. The *collinearity* condition (9) limits the offset between two contour fragments. Two similarly oriented but not collinear contour fragments will not satisfy (9). The *constant-curvature* conditions (10)–(13) assure that close to the gap the two contour fragments will have similar shapes. The probabilistic allocation algorithm always keeps the number of contour fragments within a receptive field at a minimum and thus the number of combinations to be considered by the gap bridging algorithm does not increase exponentially with the level of the pyramid. Pruning of the unprobable pairs further decreases the amount of computation.

If more than one pair of contour fragments satisfy the collinearity and constant-curvature conditions, the pair whose gap is bridged is chosen by the rule

$$\min_G \left[d \cdot \min\left(\Delta_1, 0.25(\Delta_2 + \Delta_3 + \Delta_4 + \Delta_5)\right)\right] < \frac{\pi}{4 + 2l} \tag{14}$$

where G is the ensemble of contour fragment pairs retained, d is the Euclidean distance between C_{11} and C_{21}, and l is the level in the pyramid. Rule (14) decides between a linearity measure and the average curvature around the gap. Weighting by the centroid distance d is necessary for biasing the bridging toward the closest contour fragment pair. The chosen pair must also satisfy a stricter threshold as the processing advances toward higher pyramid levels, because for the smoother contour fragments the same rigid threshold could yield undesired connections.

The gap bridging algorithm is performed on the mesh at the parents' level. Every cell carries the coordinates of its line centroids, computed for data compression (5). Conditions (9)–(13) are then verified for the contour fragment pairs located in *adjacent* cells or *within* the same cell. The different possible local configurations make both cases necessary. Two contour fragments located in adjacent cells at level l may not qualify for gap bridging, while at level $l + 1$ when seen by the same cell they can be bridged. The two contour fragments selected are connected through generating pointers to fuse their doubly linked lists. Gap bridging appears in the drawing of a low resolution representation as connecting with a line segment the two centroids closest to the gap.

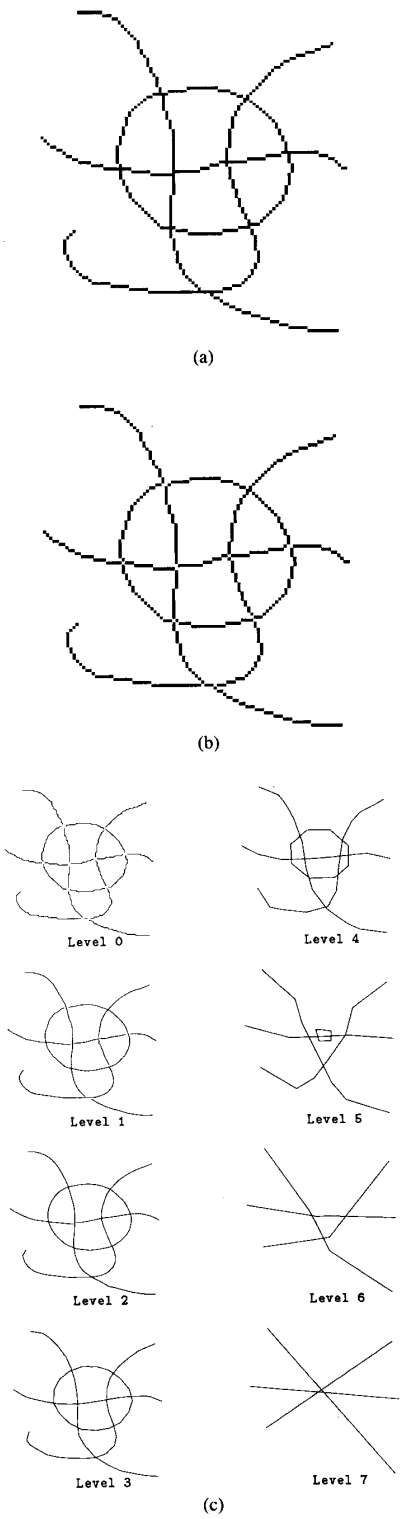


Fig. 7. Gap bridging in the chain pyramid. (a) The input image. (b) The image after removal of the branch points. (c) Representations of the contour at different levels of the chain pyramid. The circle is no longer drawn after level 5 when it has reached its root cell.

The endpoints of the contour fragments have an important but implicit role in the gap bridging algorithm. They are "pinned down," i.e., no longer allowed to shift toward the centers of the contours in the lower resolution representations as they did for smoothing (see Figs. 3 and 4). In the probabilistic allocation algorithm combined with gap bridging, the endpoints are automatically labeled 1, and thus will not be eliminated. Their parents always compute the centroids for the contour fragments within the receptive fields. This precaution is necessary to avoid the endpoints of two contour fragments moving away and widening the gap at higher levels of the chain pyramid. The perceptual quality of the interpolation is improved by the additional smoothing on the mesh after connecting the contour fragments. The smoothing is identical to the one described in the previous section. The computational formulas are given by (7).

In Fig. 7(c) the results of the gap bridging algorithm applied to the contour in Fig. 7(b) are shown at different levels of the chain pyramid. In the level 3 representation all the gaps in the input image have been connected. At higher levels, the circle, represented now by a single circular doubly linked list, shrinks because of repeated applications of the smoothing algorithm. At level 6 the root cell of the circle is reached and the contour is no longer drawn. The linear contour fragments, however, remain "visible," showing their different, more elongated nature. The phenomenon of instantaneous discrimination of a shape against a different background is known in the psychophysical literature as a pop-out phenomenon [28], [31]. The chain pyramid thus offers a simple method of modeling pop-out phenomena involving contours.

IV. Preprocessing Module

The algorithms described in the previous sections require a "clean" input, i.e., most of the pixels belonging to the contour must have only two neighbors of the same color. Contours derived from real scene images, however, tend to be several pixels wide and to contain complex branch structures. To allow application of the chain pyramid to real images a preprocessing module must generate a suitable input from the raw data.

While sophisticated contour extraction methods incorporating tracking loops based on gray level information are available [5], for our purpose the simplest approach suffices. First, an edge detector is applied to the image and the response magnitude is thresholded. The resulting binary image is then thinned by a parallel algorithm and the artifacts of thinning are eliminated by 4-connected chaining. The thinned binary image is smoothed by a local operation. At the next step, the branch points are removed as described in Section III-C and the input image of the chain pyramid is obtained. All the procedures are performed in parallel on the mesh containing the input image, i.e., level 0 of the chain pyramid.

There are no special requirements on the edge detector to be employed for obtaining the raw edge image of the input. Because only the magnitude of the edge detector

output is necessary, fast (two-mask) differential operators are recommended. For our experiments we used the 3×3 Sobel operator. The edge image is thresholded to obtain the binary image necessary for thinning. We employed manual thresholding, i.e., the optimal threshold value was chosen by inspecting the resulting binary images. The edge image should be slightly underthresholded, to allow more degrees of freedom for the thinning algorithm. The value of the threshold is not crucial; in our experiments it could vary over a range of at least ± 8 gray level values without significant change in the output of the preprocessing module (the input of the chain pyramid). Local thresholding methods, as well as adaptive procedures, are known (see [29] for a review) and can be used if the specific application requires an on-line implementation.

Thinning of the binary image is achieved by a modified version of the parallel thinning algorithm proposed by Chin *et al.* [6]. The thinning is based on 3×3 neighborhoods. The center pixel recognizes that one of the deletion or restoration templates is present in the neighborhood, and accordingly deletes itself or remains unmodified. The deletion templates are shown in Fig. 8(a). The contour pixels are indicated by 1's. Note that because of the "don't care" pixels, denoted by X's, the eight templates represent many possible configurations. The center pixel changes its value from 1 to 0 whenever a deletion template is recognized.

Most of the features in the binary images derived from edge data are elongated along one direction. The specific nature of these images is taken into account in the restoration templates [Fig. 8(b)] which differ from the ones proposed by Chin *et al.* When a restoration template is recognized in the 3×3 neighborhood the center pixel remains 1. The thinning procedure evolves synchronously and in parallel. A maximum of three iterations suffices to reach the final thinned configuration.

The underlying square lattice of the input image introduces artifacts in any thinning operation. More specifically, contours oriented along 45 degrees remain two pixels wide. To become chain-codable these contours must be further processed. The ambiguity is eliminated if instead of 8-connectivity (all the pixels in the 3×3 neighborhood are candidates for linking), 4-connectivity (only the neighbors along the vertical and horizontal are taken into account) is employed. Fig. 9 shows the result of the procedure. Thus in the branch point removal procedure, whenever more than two neighbors are counted under 8-connectivity, the cells tries to remove the uncertainty by applying 4-connectivity. If it succeeds, pointers are created and the doubly linked representation loaded into the cell. If it fails, the branch point procedure is continued as described in Section III-C.

By applying the gap bridging algorithm (Section III-C) the chain pyramid fuses the short thinned edge fragments into more meaningful features. The staircase-like con-

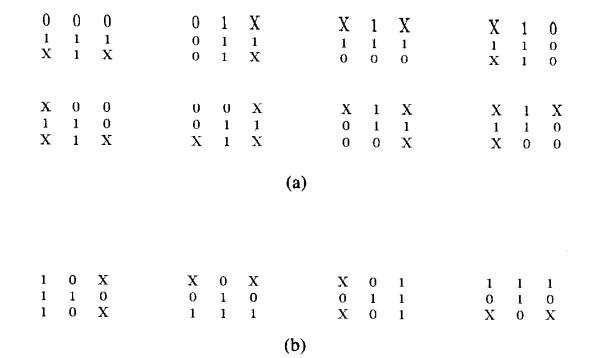


Fig. 8. Templates employed in the parallel thinning algorithm. (a) Deletion templates. (b) Restoration templates. The pixels belonging to the contour are denoted by 1's. The symbol X means that the pixel value is of no importance.

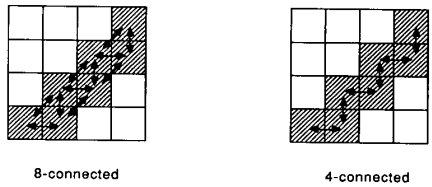


Fig. 9. Connectivity in ambiguous contours. If 8-connectivity is employed (left) all the pixels have four neighbors. Under 4-connectivity (right) the number of valid neighbors decreases to two. The arrows represent the pointers which can be created under the given connectivity rule.

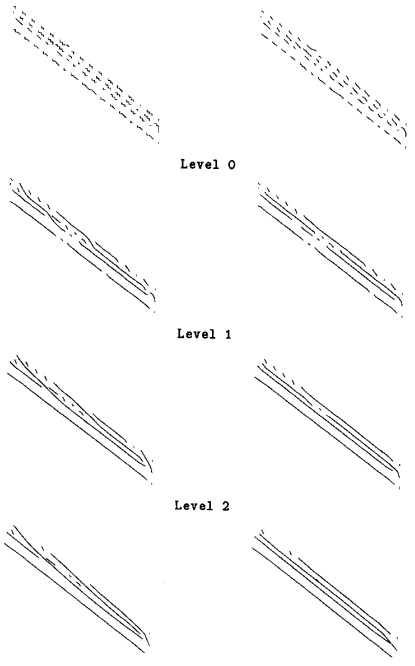


Fig. 10. The importance of smoothing on the mesh of level 0. *Left column:* Results of the gap bridging algorithm without smoothing. *Right column:* Results with smoothing.

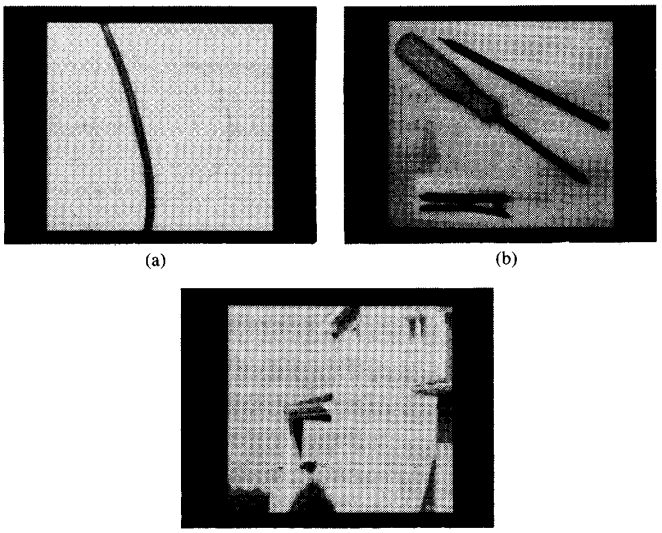


Fig. 11. Gray level images employed in the experiments. (a) *Wires*. (b) *Parts*. (c) *House*. All the images were magnified to 512 × 512 for display.

tours created by oblique edges (Fig. 9), however, are not suitable inputs into the gap bridging algorithm. In the left column of Fig. 10, the results obtained from the first three levels of the chain pyramid are shown, for a detail of the *parts* image [Fig. 11(b)]. The edge fragments (level 0) are located along parallel oblique directions and most of them were chained under 4-connectivity. These staircase contours do not group correctly and the bridging does not reproduce the parallelism of the guiding directions.

The results are drastically improved if the gap bridging procedure starts by smoothing along the contour on the mesh of level 0. The effect of smoothing can be assessed by comparing the two representations of level 0 in Fig. 10. The smoothing procedure is the one described in Section III-B and involves the two neighbors along the contour. In most of the cases the gaps are bridged correctly on the subsequent chain pyramid levels and the long parallel features are recovered.

We have applied the chain pyramid to three gray level images shown in Fig. 11. The *wires* [Fig. 11(a)] and *parts* [Fig. 11(b)] images are of size 256 × 256, and the *house* image [Fig. 11(c)] is 128 × 128. The results are shown in Figs. 12–14. Because of excessive smoothing and bridging, at higher levels of the chain pyramid (above level 3) the representations tend to become too coarse. This artifact can, however, be useful if only the principal directions of the image are sought. At the highest levels of the chain pyramid most of the contour fragments will group along these directions.

The long parallel edge in the *wires* image (Fig. 12) are kept separated in spite of their small separations. While a few nondesired bridgings exist, especially at the crossings

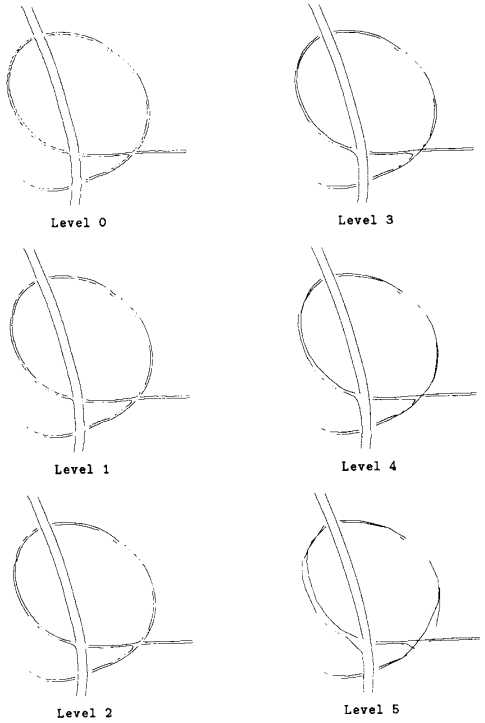


Fig. 12. Results of the gap bridging algorithm for the *wires* image.

of the wires, these can be eliminated by incorporating *a priori* information about the objects into the processing. Recall that only bottom-up computations are performed by the chain pyramid in the algorithms described here.

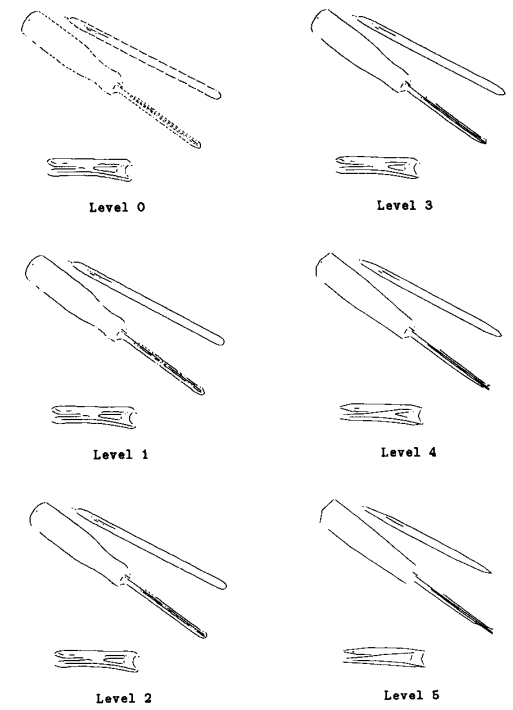


Fig. 13. Results of the gap bridging algorithm for the *parts* image.

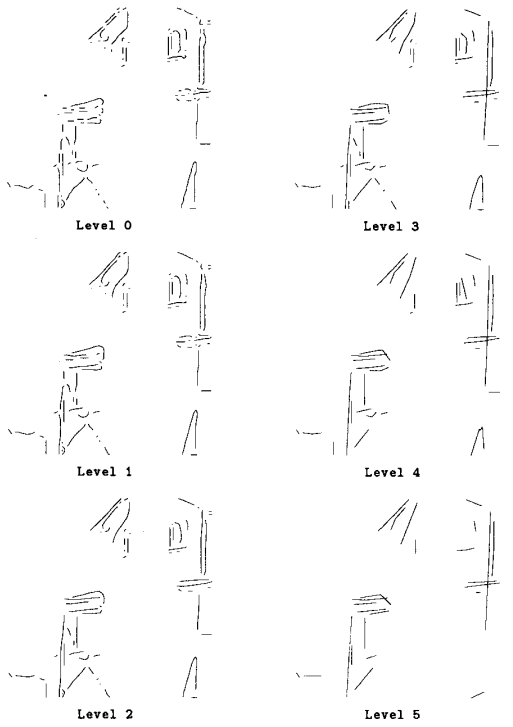


Fig. 14. Results of the gap bridging algorithm for the *house* image.

The shape of the objects in the *parts* image (Fig. 13) can be recovered from the level 2 or level 3 chain pyramid representation. At these levels most of the small curve fragments have been fused and the smoothing has not yet taken its toll. The doubly linked lists carrying the features can be input into pattern recognition and/or reasoning modules. The *house* image (Fig. 14) is the most complex example. The main features are extracted at level 2 and the principal directions of scene appear at levels 4 and 5.

V. Summary and Further Directions of Research

To conclude, the following sequence of operations is applied to a gray level image when it is processed in the chain pyramid:

- Edge detection and magnitude thresholding. A binary image is obtained.
- Parallel thinning. Generation of the level 0 doubly linked list representation.

Includes:

—elimination of connection ambiguities;
—branch point removal.

- Smoothing on the mesh of level 0.
- Application of the gap bridging algorithm. Involves at every level:

—establishing local connectivity;
—probabilistic allocation;
—data compression;
—choosing the pair of contour fragments to be bridged (if any);
—smoothing the new representation.

Processing in the chain pyramid is modular. The gap bridging algorithm discussed in detail above is a good example of this modularity. New applications can be implemented without any change in the already existing modules. An example of such an application could be the *segmentation of contours*. The partitioning of the contours into perceptually significant pieces is of importance for computer vision [11]. Several parallel algorithms for contour partitioning are available (see [30] for a comparative study) and they can easily be implemented on the chain pyramid.

Usually the goal in extracting contours from real images is to generate a suitable input for a high-level vision module which can interpret the resulting line drawing. The interpretation is always task oriented and the available *a priori* information can already be used in the algorithms extracting the contours. The processes are no longer bottom-up only; an important top-down component is also present. Systems have been described by Fua and Hanson [10] for aerial image delineation, by Reynolds and Beveridge [26] for natural scenes, and by Darwish and Jain [7] for printed circuit boards. Many of the rules employed by these systems are based on local contour configurations and thus can be implemented in the parallel hierarchical environment of the chain pyramid, yielding significant speed-ups.

The chain pyramid employs the original definition of the chain code [12]. Recently the subject has received renewed attention in the literature and the proposed improvements [13], [17], [23] can be implemented in a hierarchical environment similar to the one described in this paper.

We have presented a set of algorithms, implemented in the hierarchical environment of the chain pyramid, for fast parallel processing of contours. Artifacts caused by the rigid structure of the image pyramid were eliminated by a probabilistic allocation algorithm. Different operations on the chain pyramid can be combined modularly, and further extensions are possible.

REFERENCES

[1] I. Agi, P. J. Hurst, and A. K. Jain, "An expandable VLSI processor array approach to contour tracing," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, New York, Apr. 11-15, 1988, pp. 1969-1972.

[2] H. Asada and M. Brady, "The curvature primal sketch," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 2-14, 1986.

[3] P. J. Burt, "Fast filter transforms for image processing," *Comput. Graphics Image Processing*, vol. 16, pp. 20-51, 1981.

[4] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Trans. Graphics*, vol. 2, pp. 217-236, 1983.

[5] B. D. Chen and P. Siy, "Forward/backward contour tracing with feedback," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, pp. 438-446, 1987.

[6] R. T. Chin, H. K. Wan, D. L. Stover, and R. D. Iverson, "A one-pass thinning algorithm and its parallel implementation," *Comput. Vision, Graphics, Image Processing*, vol. 40, pp. 30-40, 1987.

[7] A. M. Darwish and A. K. Jain, "A rule based approach for visual pattern inspection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-10, pp. 56-68, 1988.

[8] I. Dinstein, D. W. L. Yen, and M. D. Flickner, "Handling memory overflow in connected component labeling applications," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 116-121, 1985.

[9] S. Edelman, "Line connectivity algorithms for an asynchronous pyramid computer," *Comput. Vision Graphics Image Processing*, vol. 24, pp. 169-187, 1987.

[10] P. Fua and A. J. Hanson, "Using generic geometric models for intelligent shape extraction," in *Proc. Image Understanding Workshop*, Los Angeles, CA, Feb. 23-25, 1987. Los Altos, CA: Morgan Kaufmann, 1987, pp. 227-233.

[11] M. A. Fischler and R. C. Bolles, "Perceptual organization and curve partitioning," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 100-105, 1986.

[12] H. Freeman, "Computer processing of line-drawing images," *Comput. Surveys*, vol. 6, pp. 57-97, 1974.

[13] L. O'Gorman, "Primitives chain codes," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, New York, Apr. 11-15, 1988, pp. 792-795.

[14] G. Hartmann, "Recognition of hierarchically encoded images by technical and biological systems," *Biol. Cybern.*, vol. 57, pp. 73-84, 1987.

[15] T. S. Hong, M. O. Shneier, R. L. Harley, and A. Rosenfeld, "Using pyramids to detect good continuation," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-13, pp. 631-635, 1983.

[16] E. Horowitz and S. Sahni, *Fundamentals of Data Structures*. Rockville, MD: Computer Science Press, 1976.

[17] J. Koplowitz and A. P. S. Raj, "A robust filtering algorithm for subpixel reconstruction of chain coded line drawings," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, pp. 451-456, 1987.

[18] W. G. Kropatsch, "Curve representations in multiple resolutions," *Pattern Recogn. Lett.*, vol. 6, pp. 179-184, 1988.

[19] J. V. Mahoney, "Image chunking: Defining spatial building blocks for scene analysis," Artificial Intell. Lab., Massachusetts Inst. Technol., Rep. AI-TR-980, Aug. 1987.

[20] P. Meer, "Simulation of constant size multiresolution representations on image pyramids," *Pattern Recogn. Lett.*, vol. 8, pp. 229-236, 1988.

[21] ——, "Stochastic image pyramids," *Comput. Vision, Graphics, Image Processing*, vol. 45, pp. 269-294, 1989.

[22] R. Miller and Q. S. Stout, "Data movement techniques for the pyramid computer," *SIAM J. Comput.*, vol. 16, pp. 38-60, 1987.

[23] T. Minami and K. Shinohara, "Encoding line drawings with multiple grid chain code," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 269-276, 1986.

[24] F. Mokhtarian and A. Mackworth, "Scale-based description and recognition of planar curves and two-dimensional objects," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 34-43, 1986.

[25] F. P. Preparata and M. I. Shamos, *Computational Geometry. An Introduction*. Berlin: Springer, 1985.

[26] G. Reynolds and J. R. Beveridge, "Searching for geometric structures in images of natural scenes," in *Proc. Image Understanding Workshop*, Los Angeles, CA, Feb. 23-25, 1987. Los Altos, CA: Morgan Kaufmann, 1987, pp. 257-271.

[27] W. Richards, B. Dawson, and D. Whittington, "Encoding contour shape by curvature extrema," *J. Opt. Soc. Amer. A*, vol. 3, pp. 1483-1491, 1986.

[28] A. Rosenfeld, "Recognizing unexpected objects: A proposed approach," *Int. J. Pattern Recogn. Artificial Intell.*, vol. 1, pp. 71-84, 1987.

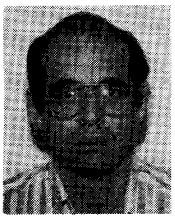
[29] P. K. Sahoo, S. Soltani, and A. K. C. Wong, "A survey of thresholding techniques," *Comput. Vision, Graphics, Image Processing*, vol. 41, pp. 233-260, 1988.

[30] C. H. Teh and R. T. Chin, "A scale-independent dominant point detection algorithm," in *Proc. IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition*, Ann Arbor, MI, June 5-9, 1988, pp. 229-234.

[31] A. Treisman, "Preattentive processing in vision," *Comput. Vision, Graphics, Image Processing*, vol. 31, pp. 156-177, 1985.

[32] D. L. Tuomenoksa, G. B. Adams, H. J. Siegel, and O. R. Mitchell, "A parallel algorithm for contour extraction: Advantages and architectural implications," in *Proc. IEEE Comput. Soc. Conf. Computer Vision and Pattern Recognition*, Washington, DC, June 19-23, 1983, pp. 336-344.

[33] S. Ullman, "Visual routines," *Cognition*, vol. 18, pp. 97-159, 1984.

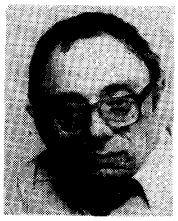


Peter Meer (S'84-M'86) was born in Oradea, Romania, on February 14, 1949. He received the Dipl. Engn. degree from the Bucharest Polytechnic Institute, Bucharest, Romania, in 1971, and the D.Sc. degree from the Technion, Israel Institute of Technology, Haifa, Israel, in 1986, both in electrical engineering.

From 1971 to 1979 he was with the Computer Research Institute, Cluj, Romania, working on R&D of digital hardware. From 1980 to 1986 he was with the Technion, doing research on computational models of human vision. Since 1986 he has been with the Center for Automation Research, University of Maryland, College Park, currently as Assistant Research Scientist. He is author of over 20 technical papers. His fields of interests are in human and machine vision, digital signal processing, and parallel algorithms.

Chiao-Yung Allen Sher received the B.S. degree in electrical engineering from National Taiwan University in 1981 and the M.S. degree in computer science from the University of Maryland, College Park, in 1988.

Currently, he is a doctoral student in computer science at the University of Maryland, College Park, and a research assistant at the Computer Vision Laboratory of the University's Center for Automation Research. His research interests include computer vision and parallel algorithms.



Azriel Rosenfeld (M'60-F'72) was born in New York City on February 19, 1931. He received rabbinic ordination in 1952, the Doctor of Hebrew Literature degree from Yeshiva University, New York, NY, in 1955, and the Ph.D. degree in mathematics from Columbia University, New York, NY, in 1957.

He is a tenured Research Professor and Director of the Center for Automation Research, a department-level unit of the University of Maryland in College Park. He also holds Affiliate Professorships in the Departments of Computer Science and Psychology and in the College of Engineering. He is a widely known researcher in the field of computer image analysis. He wrote the first textbook in the field (1969); was a founding editor of its first journal (1972); and was co-chairman of its first international conference (1987). He has published over 20 books and over 400 book chapters and journal articles, and has directed over 30 Ph.D. dissertations.

Dr. Rosenfeld, won the IEEE Emanuel Piore Award in 1985; he was a founding Director of the Machine Vision Association of the Society of Manufacturing Engineers (1985-1988), and won its President's Award in 1987; he was a founding member of the IEEE Computer Society's Technical Committee on Pattern Analysis and Machine Intelligence (1965), served as its Chairman (1985-1987), and received the Society's Meritorious Service Award in 1986; he was a founding member of the Governing Board of the International Association for Pattern Recognition (1978-1985), served as its President (1980-1982), and won its first K.S. Fu Award in 1988; he is a Fellow of the Washington Academy of Sciences (1988), and won its Mathematics and Computer Science Award in 1988; he is a Corresponding Member of the National Academy of Engineering of Mexico (1982) and a Foreign Member of the Academy of Science of the German Democratic Republic (1988). He holds an honorary Doctor of Technology degree from Linköping University, Sweden (1980) and is a certified Manufacturing Engineer (1988).