Polygonal Approximation And Scale-Space Analysis Of Closed Digital Curves

Author: Bimal Kumar Ray

This thesis is submitted to Indian Statistical Institute, Calcutta, in partial fulfillment of the requirements for the award of Ph. D. degree.

Calcutta

1994

Contents

Chapter 1

Introduction

This thesis presents a series of algorithms for polygonal approximation of closed digital curves followed by scale-space analysis with its application to corner detection.

Approximation of a closed curve by piece straight line segments is known as polygonal approximation. Any curve can be approximated by a polygon with any desired degree of accuracy.

Polygonal approximation is useful in reducing the number of points required to represent a curve and to smooth data. Such representation facilitates extraction of numerical features for description and classification of curves.

Basically there are two approaches to the problem. One is to subdivide the points into groups each of which satisfies a specific criterion function measuring the collinearity of the points. The collinearity is measured either by integral square error or by absolute error or by some other criterion function such as area deviation. This approach treats polygonal approximation as a side detection problem. Another approach to polygonal approximation is to detect the significant points and join the adjacent significant points by straight line segments. This approach treats polygonal approximation as an angle detection problem. In this thesis we treat polygonal approximation as a side detection as well as an angle detection problem.

A number of algorithms for polygonal approximation of digital curves are already existing. Ramer [38] and Duda and Hart [13] propose a splitting technique which iteratively splits a curve into smaller and smaller curve segments until the maximum

of the perpendicular distances of the points of the curve segment from the line joining the initium and the terminus of the curve segment falls within a specified tolerance. The curve segments are split at the point most distant from the line segment. The polygon is obtained joining the adjacent break-points. The procedure need multiple passes through data. Pavlidis and Horowitz [33] use split-and-merge technique which fits lines to an initial segmentation of the boundary points and computes the least squares error. The procedure then iteratively splits a curve if the error is too large and merges two lines if the error is too small. Pavlidis [32] develops another procedure which is based on the concept of 'almost collinearity' of a set of points. To check whether a set of points are collinear / almost collinear the procedure computes an error of fit which is a function of two variables C and T. T is the maximum of the perpendicular distances of the points being tested for collinearity from the yet-to-be obtained line segment. And C is a normalized variable ($0 \leq C \leq 1$) which is determined by the ratio of the number of sign changes to the total number of sign changes the perpendicular distance goes through. If $T - CW_0 - T_0 < 0$ then the line segment is acceptable, else if $T - CW_0 - T_0 > 0$ then the line segment is rejected, T_0 being the acceptable error and W_0 is the weighting factor of C. The procedure need multiple passes through data.

The fundamental problem in the splitting technique and in the split-and-merge process is the initial segmentation. For open curves one can start with the two end points of a curve as the initial break-points. If a curve be closed Ramer [38] suggests that the top left most point and the bottom right most point can be taken as the initial break-points. In the split-and-merge process the initial segmentation is done by decomposing the curve into a suitable number of points. Unfortunately, in this approach the approximation depends on the initial segmentation. A different mode of initial segmentation generates a different approximation. This difficulty is caused by the arbitrary initial segmentation. We believe that if the initial segmentation is done following some deterministic rules then this difficulty may be overcome.

Ansari and Delp [3] make the initial segmentation using the extreme curvature (maximum of positive curvature and minimum of negative curvature) points as the initial set of break-points. They convolve the data points with the Gaussian kernel and find the extreme curvature points of the Gaussian smoothed curve. Those points

along the original boundary which correspond to the extreme curvature points are used as the initial set of break-points.

In chapter 2 we introduce the concept of *rank* of a point to resolve the problem of initial segmentation. The split-and-merge is done using the perpendicular distance of a point from line segment joining the initium and the terminus of a curve segment. The initial segmentation is computationally less expensive as compared to that in [3].

In chapter 3 we introduce a sequential one-pass algorithm [40]. A number of sequential one-pass algorithms for polygonal approximation of digital curves are already existing. One such algorithm proposed by Williams [59] uses cone intersection method to find the maximal possible line segments. Circles of specified radii are drawn around each point until the intersection of the cones with their vertex at the initial point and touching the circles is an empty set. The segments are obtained by joining the initial point to the last point that passed the test. The procedure need a single pass through data.

Wall and Danielsson [58] develop another sequential one-pass technique which is based on the concept of area deviation per unit length of the approximating line segment. The procedure finds the maximal line segment by merging points one after another with the initial point until the area deviation per unit length of the approximating line segment exceeds a maximum allowed value. The line segment is obtained joining the initial point to the last point that passed the test.

Though Williams' procedure [59] is sequential one-pass and it need a small and finite memory (so the procedure is highly efficient) but it has two defects. Firstly, it need a considerably large number of arithmetic operations and secondly, it misses corners and rounds off sharp turnings (spikes). The algorithm designed by Wall and Danielsson [58] is faster than that of Williams'. Williams' procedure need seven multiplications / divisions and evaluation of one square root, whereas, Wall and Danielsson's procedure need six multiplications and the simplified version of it need only three to four multiplications. For chain coded curves Williams' procedure need five multiplications, whereas, Wall and Danielsson's procedure need two multiplications and its simplified version only one. The basic version of Wall and Danielsson's procedure, too, rounds off sharp turnings. They introduced a "peak

test" which retained the sharp turnings.

The sequential one-pass algorithm [40] that we propose in chapter 3 need no arithmetics except subtractions. The procedure is based on a proposition from numerical analysis and some concepts from regression analysis. The vertices are located by identifying some patterns exhibited by the first order finite differences of the boundary point data. Since the procedure need no arithmetic except subtractions hence its computational load is very much lower than that of the existing sequential techniques and neither does it round off sharp turnings nor does it miss corners.

Though this technique is computationally simpler than the existing ones but it holds for curves with uniformly spaced points only. In chapter 4 we present another sequential one-pass algorithm [41] which holds for curves with uniformly spaced points as well as non-uniformly spaced points and it is computationally less expensive than the other sequential one-pass algorithms. The procedure is based on Pavlidis' concept [32] of almost collinearity of a sequence of points. The collinearity is checked by measuring the area and the perimeter of the triangle formed by sequence of points triplets.

In the polygonal approximation techniques discussed so far the maximum allowable error is specified either directly or indirectly. Stone [55] considers the problem of approximating a known non-linear function by a polygon with a specified number of line segments. The procedure minimizes the sum of squares of errors between the known function and the line segments forming the polygon and thereby determines the points of subdivision and the parameter of the line segments. Stone's approach is classical in nature. Bellman [9] use dynamic programming [8] to solve the same problem. Bellman primarily confines himself to the analytical aspects of the solution, briefly mentioning how the solution of the equation for each particular point of subdivision can be reduced to a discrete search. He further suggests the extension of his method to polynomial fittings to known functions. Gluss has written a series of papers [19], [20], [21]. In [19] Gluss considers the computational aspects of Bellman's work fully, noting the similarities to some of Stone's equations and deduces an equation to determine the points of subdivision that involves an equality rather than minimization. Stone's procedure does not necessarily produce

continuous approximation. The line segments can be, and in general will be broken in the sense that they need not meet at the points of subdivision. Only when the given function is quadratic, Stone obtains a continuous approximation. Bellman's procedure, too, does not necessarily produce continuous approximation. In [19] Gluss considers a model in which the lines are constrained to meet on the curve at the points of subdivision. In [20] Gluss considers the problem of approximating a continuous non-linear function by a polygon where the line segments are considered to meet at the point of subdivision but not necessarily on the curve. The method of solution presented in [20] involves a functional of two variables. This makes the solution much more cumbersome than for the same problem without continuity constraint which involves a functional of a single variable [9], [19]. In [21] Gluss avoids this difficulty by introducing a new criterion function which involves derivative of known function and the slope of the line segments. Cantoni [12] solves the problem of finding a continuous polygonal approximation of a known non-linear function by minimizing the weighted integral square error (called the performance index) between the known function and the approximating line segments. The approach is classical in nature. Pavlidis [34] obtains approximation of planar curves and waveforms using integral square error as the criterion function. The break-points are located by finding the zero of the first order derivative of integral square error by applying Newton's vector method.

All these algorithms produce optimal polygon in the least squares sense. In all these works the user of the procedure has to specify the number of line segments and minimize the sum of square of errors. Dunham [14] suggests an optimal algorithm which, instead of specifying the number of line segments, specifies the error (L_∞ norm) and determines the minimum number of line segments. Dynamic programming is used to solve the problem. The recurrence relation used to determine the minimum number of line segments is simple. A scan along implementation of the algorithm is made along the lines of Williams [59] and Skalansky and Gonzalez [54]. The approximation is continuous and the knot points are constrained to lie on the curve. (Some other works on polygonal / piecewise linear approximation are cited in the bibliography.)

In all these algorithms the user of the procedure has to specify either the number

of line segments or the maximum allowable error. The maximum allowable error or the number of line segments to be used are generally determined on the basis of a trial and error process. So these procedures cannot run without operator's intervention.

In chapter 5 we are looking for a data-driven method [42] in which neither do we specify the error nor do we specify the number of line segments. We keep both these parameters free and allow the procedure to determine them on the basis of the local topography of the curve. The procedure looks for the longest possible line segment with the minimum possible error. Integral square error is selected as the error norm to measure the closeness of the polygon to the digital curve. An objective function is constructed using the length of a line segment joining two points of the curve and the integral square error along the line segment. The vertices are located at the points where this objective function attains local maxima. The local maxima are looked for by a discrete search method. The objective function is computed by merging points one after with an arbitrarily selected initial point until a local maxima is found. The process is restarted from the new vertex and is repeated and carried beyond the starting point till the vertex generated last coincides with one of the vertices already generated. The procedure is sequential and one-pass but neither does it miss corner points nor does it round off sharp turnings (spikes). Moreover, the number of arithmetic operations required by this scheme is less than that required by the Williams' scheme. Williams' scheme need seven multiplications / divisions and extraction of one square root whereas, this procedure need only five multiplications / divisions and extraction of one square root. Though the number of arithmetic operations required by the Wall and Danielsson's scheme is comparable to that required by this procedure, but the former cannot run without operator's intervention.

All algorithms for polygonal approximation use either the integral square error or the absolute error as a measure of closeness. In chapter 6 we present another sequential one-pass algorithm [43] where the sum of absolute errors (L_1 norm) is used as a measure of closeness. The procedure is technically and conceptually similar to that we present in chapter 5. The objective of designing this scheme is to show that: though the most commonly used norms are integral square error and the

absolute error but it is possible to use L_1 norm to make polygonal approximation of digital curves.

The procedures over viewed so far treat polygonal approximation as a side detection problem. The sides of a polygon are determined subject to certain constraints on the goodness of fit. Another approach to polygonal approximation is to detect the significant points and join the adjacent significant points by straight line segments. Significant points are of two types, namely, curvature extrema points and points of inflexion. The concept of detecting local curvature extrema originates from Attneave's famous observation [6] that information about a curve is concentrated at the curvature extrema points. Freeman [18] suggests that the points of inflexion carry information about a curve and so these points can be used as significant points.

A series of algorithms on the detection of significant points on digital curves are already existing. Rosenfeld and Johnston [48], in an attempt to determine whether a procedure [49] designed to detect discontinuities in the average gray level can also detect discontinuities in the average slope, detect significant points as the curvature extrema points. The procedure is parallel and need an input parameter m. The value of m is taken to be 1/10 or 1/15 of the perimeter of the curve. The input parameter is introduced to determine the region of support and the $k-$cosine of the boundary points. The significant points are the local maxima of $k-$cosine.

An improved version of this procedure is given by Rosenfeld and Weszka [50]. They use smoothed $k-$cosine to determine the region of support and to detect the significant points. The procedure is parallel and need an input parameter m as in [48].

Freeman and Davis [17] design a corner-finding scheme which detects local maxima of curvature as significant points. The algorithm consists of scanning the chain code of a curve with a moving line segment which connects the end points of a sequence of s links. As the line segment moves from one chain node to the next, the angular difference between the successive segment positions are used as a smoothed measure of local curvature along the chain. The procedure is parallel and need two input parameters s and m. Both are smoothing parameters and their assigned values determine the degree of smoothing. The greater the s, the heavier is the

smoothing. The parameter m is used to allow some stray noise. For a well quantized chain s will always be a relatively small number ranging from 5 to 13. And the parameter m will take value either 1 or 2.

Anderson and Bezdek [2] devise a vertex detection algorithm which, instead of approximating discrete curvature, defines tangential deflection and curvature of discrete curves on the basis of the geometrical and statistical properties associated with the eigenvalue-eigenvector structure of sample covariance matrices. The vertices are the significant points in the sense that they carry information about the curve. The procedure is sequential and need more than one input parameter.

Sankar and Sharma [53] design an iterative procedure to detect significant points as points of maximum global curvature based on the local curvature of each point with respect to its immediate neighbors. The procedure is parallel. In contrast to the algorithms [48], [50], [17] and [2], it does not need any input parameter.

Each of the algorithms [48], [50], [17] and [2] need one or more input parameters. The choice of these parameters is primarily based on the level of detail of the curves. In general, it is difficult to choose a set of parameters that can successfully be used to detect the significant points of a curve which consists of features of multiple size. Too large a parameter will smooth out fine features, and too small a parameter will generate a large number of unwanted significant points. This is the fundamental problem of scale, because the features describing the shape of a curve vary enormously in size and extent, and there is seldom any basis of choosing a particular value of parameter for a particular feature size [61]. Though Sankar-Sharma's scheme [53] does not need any input parameter, it does not involve determination of region of support. The procedure is iterative in nature and fails to operate successfully on curves which consist of features of multiple size.

To detect dominant points (curvature maxima points) Teh and Chin [56] have designed a procedure which need no input parameter and remains reliable even when features of multiple size are present on the curve. In contrast to the existing belief that the detection of dominant points depends heavily on the accurate measures of significance (e.g. $k-$cosine, $k-$curvature, cornerity measure, weighted curvature measure), Teh and Chin make an important observation: The detection of dominant points relies not only on the accuracy of the measures of significance, but

primarily on the precise determination of the region of support. Their procedure is motivated by the Rosenfeld-Johnston angle detection algorithm [48], in which both an incorrect region of support and an incorrect curvature measure may be assigned to a point if the input parameter is not chosen correctly, and hence dominant points may be suppressed [15]. To overcome this problem they propose that the region of support and hence the corresponding scale factor or the smoothing parameter of each boundary point should be determined independently, based on its local properties. They use chord length and perpendicular distance to determine the region of support and have further shown that once the region of support of each point is determined, various measures of significance can be computed accurately for detection of dominant points. The dominant points are detected as the local maxima of k—cosine, k—curvature and 1-curvature. All these measures of significance are found to produce almost the same results. The procedure is parallel and need no input parameter.

In chapter 7 we present a new algorithm [44] for detection of significant points of digital curves. The procedure is motivated by the Rosenfeld-Johnston angle detection scheme [48]. The procedure need no input parameter and remains reliable even when features of multiple size are present. The k—cosine is used to determine the region of support. A new measure of significance based on k—cosine (smoothed k—cosine) is introduced. The significant points are the local maxima / minima of the smoothed k—cosine. The procedure is parallel. The objective of this work is to show that one can use k—cosine itself to determine the region of support without using any input parameter.

In the works on the detection of significant points the region of support (if introduced) consists of equal number of points on either side of the point of interest. We propose to call this region of support as *symmetric* region of support. But the local properties of a curve may not everywhere be so as to have a symmetric region of support. We believe that an *asymmetric* region of support consisting of unequal number of points on either side of the point of interest is more natural and more reasonable than a symmetric region of support. A symmetric region of support may be looked upon as a special case of asymmetric region of support. In chapter 8 we present a new technique [45] on the detection of dominant points (curvature maxima

points) which, unlike the existing algorithms, introduces the concept of *asymmetric* region of support based on the local properties of a curve. A *new* measure of significance, called $k-l$ cosine, is introduced. The dominant points are the local maxima of $k-l$ cosine. The procedure is parallel. It need no input parameter and remains reliable even when features of multiple size are present on the curve.

As already stated the fundamental problem in data smoothing is choice of input parameter. Too large a parameter will smooth out many important features (corners, vertices, zero-crossings) and too small a parameter will produce many redundant features. This is the fundamental problem of scale because, features appearing on a curve vary enormously in size and extent and there is seldom any basis of choosing a particular parameter for a feature of particular size. This problem may be resolved by automatic parameter tuning. The parameter size can be tuned on the basis of the local properties of the curve using a suitable criterion function. Teh and Chin [56] resolve this problem using chord length and perpendicular distance as the criterion function to determine the region of support of each point on the basis of the local properties of the curve.

Another approach to the solution of the problem of scale (choice of input parameter) is scale-space analysis. The concept of scale-space was introduced by Witkin [61] and Koenderink and van Doorn [25]. Scale-space analysis of a signal is made by convolving it with the Gaussian kernel treating the parameter σ of the kernel as a continuous scale parameter. The zero-crossings of curvature / extreme curvature points of the smoothed signal are located by varying the parameter continuously. The arc length is shown along the $x-$ axis (horizontal) and the scale parameter along the $y-$axis (vertical). The image on the xy half plane showing the location of curvature extrema or the zero crossings of curvature at varying scales is called scale-space map.

Mokhtarian and Mackworth [31] extend this work to two-dimensional shapes locating zero-crossings of curvature over scales.

Asada and Brady [5] use the concept of scale-space filtering to extract primitives such as corners, smooth joins, crank, bump / dent from the bounding contours of planar shapes. They make scale-space analysis of the behavior of these primitives by varying the parameter of the Gaussian kernel in one octave, corresponding to

multiplying by $\sqrt{2}$. A tree representation showing the movement of the position of the local positive maxima and negative minima in the first and second derivatives of the Gaussian smoothed curve is constructed by varying the parameter of the kernel. The primitives are detected and located in a process of parsing the tree. Using the location of the primitives at each scale as a set of knot points they have made a polygonal approximation, circular spline approximation, cubic spline approximation and B-spline approximation of the planar shapes.

Saint-Marc et al. [52] suggest adaptive smoothing leading to construction of scale-space map without using the Gaussian kernel. The smoothing is done using a decaying exponential window which is a function of a smoothing parameter k and a measure of signal discontinuity. The underlying concept of the procedure is to keep the window size of the smoothing kernel constant and to apply the kernel iteratively on the signal. They construct two types of scale-space map. In one kind they keep the parameter k fixed and use the number of iterations as the scale parameter. Here the parameter k determines the magnitudes of the edges (corners) to be preserved during the smoothing process. They call this scale-space map as the Gaussian scale-space map. The other scale-space map that they construct has been referred to as the adaptive scale-space map. Here the number of iterations to be performed is held fixed and the parameter k is varied to construct the scale-space map. Without making an attempt to give a multi-scale interpretation of the map they detect corners on planer shapes for different values of k. They also show application of the procedure to edge detection from gray level image and to range image segmentation.

Meer et al. [30] suggest a method to detect dominant points by first determining the optimal scale of a Gaussian-like convolution multiple scale representation of the boundary. Then, a measure of optimality, which is directly proportional to the total curvature of the boundary, is defined. The optimal scale is determined such that the difference in the measure of optimality between two successive scales is the smallest. The corners are detected at the optimal scale. The procedure does not take into account various levels of detail of a curve.

Rattarangsi and Chin [39] use the concept of scale-space filtering to design a corner detector which takes into account various levels of detail of a curve. They make

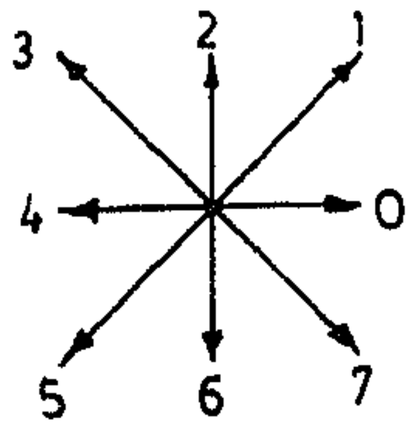


Figure 1.1: Directions in Freeman chain code

an analysis of the scale-space behavior of different corner models such as Γ models, END models and STAIR models. The extreme curvature points are detected and located by convolving the curve with the Gaussian kernel with varying window size. The scale-space map shows the movement of the extreme curvature points over scales. The scale-space map is converted into a tree representation with the help of two assumptions namely, identification and localization. A number of stability criteria are derived on the basis of the scale-space behavior of the corner models. The tree is interpreted using these stability criteria and corners are detected and located. The procedure works well on curves which consist of features of multiple size and is robust to noise.

In chapter 9 we present scale-space analysis of digital curves using one of our polygonal approximation schemes and show its application to corner detection. The procedure [46] presented in this chapter holds for curves with uniformly spaced points only. The scale-space analysis is made without convolution with a smoothing kernel. The corner detection is done without estimating curvature.

In chapter 10 and chapter 11 we present scale-space analysis using convolution with a smoothing kernel and corner detection is done via curvature estimation. In chapter 10 digital Gaussian filter coefficients are used iteratively to convolve the curve whereas in chapter 11 a discrete scale-space kernel is used.

Finally in chapter 12 we draw conclusion.

Before we close this chapter we give the definition of a closed digital curve. A closed digital curve C_d with n points is defined by a sequence of n integer coordinate points

$$C_d = \{\, p_i = (x_i\,,\, y_i),\ i = 1,\, 2, ...,\, n \,\}\,,$$

where p_{i+1} is a neighbor of p_i (modulo n).

The Freeman chain code of C_d consists of n vectors $\mathbf{c}_i = \overline{p_i p_{i+1}}$ each of which can be represented by an integer $f = 0, 1, 2, ..., 7$ as shown in Figure 1.1, where $\frac{1}{4}\pi f$ is the angle between the x-axis and the vector $\mathbf{c}_i$. The chain of C_d is defined by $\{\mathbf{c}_i,\ i = 1, 2, ..., n\}$ and $\mathbf{c}_i = \mathbf{c}_{i \pm n}$.

Chapter 2

A split-and-merge technique

The fundamental problem in the existing splitting techniques and also in the split-and-merge techniques is the initial segmentation. Ansari and Delp [3] try to resolve this problem using curvature extrema points as the initial set of breakpoints.

In this chapter we present an alternative approach to initial segmentation for chain coded curves. The initial segmentation is done introducing the concept of rank of a point defined in the following section. The split-and-merge is done using the absolute perpendicular distance of a point from the line segment joining the initium and the terminus of a curve segment as the criterion function.

2.1 Procedure

Before we present the procedure we define *rank* of a point. The smallest angle through which the vector $\mathbf{c}_i$ should be rotated so that $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ have the same directions determines the rank of the point p_i and if θ_i be the smallest required angle then the rank of p_i is defined by

$$r_i = (4/\pi)\,\theta_i\,, \qquad i = 1, 2, ..., n\,. \tag{2.1}$$

We note that the only possible values for r_i are 0, 1, 2, 3, and 4. The angle θ_i is computed by the relation

$$\theta_i = cos^{-1}\left(\frac{\mathbf{c}_i \,.\, \mathbf{c}_{i+1}}{|\,\mathbf{c}_i\,|\,|\,\mathbf{c}_{i+1}\,|}\right). \tag{2.2}$$

As already mentioned the fundamental problem in the split-and-merge process is the initial segmentation. We propose to make the initial segmentation on basis of the rank of a point. The procedure *Initial segmentation* looks for those points with rank greater than or equal to 3. If there exist at least two points with rank greater than or equal to 3 then these points are used as the initial set of break-points. If there exists only one point with rank greater than or equal to 3 then this point together with those with rank equal to 2 constitute the initial set of break-points. If there is no point with rank greater than or equal to 3 and there exist at least two points with rank 2 then these points only constitute the initial set of break-points. If there exists only one point with rank 2 then this point together with those points with rank 1 comprise the initial segmentation provided there is at least one point with rank 1. Lastly, if there exist only two points with rank 1 then one can start with these points. We note that we give priority to those points which have higher rank. This approach reduces the false choice of vertices in initial segmentation.

Since the rank is obtained by taking the angular difference between the vector $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$ and multiplying it by $4/\pi$ hence the computational load is lower than that of Ansari and Delp [3] in which the total number of multiplications and additions depends on the window length of the Gaussian filter. The larger the window length is, the higher is the computational load.

Using the initial set of break-points we perform a split-and-merge process. The split-and-merge is done by a collinearity check. The criterion function for collinearity check is the absolute perpendicular distance of a point from the line segment joining two successive break-points.

The perpendicular distance of a point p_k from the segment joining two successive break-points p_i and p_j is computed by the formula

$$d_k = \frac{|(y_j - y_i)x_k - (x_j - x_i)y_k - x_i y_j + x_j y_i|}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}}, \tag{2.3}$$

where $k = i+1, i+2, ..., j-1$. Points are split at those points where d_k is a maximum and exceeds a specified value, which we take as one pixel. This threshold of one pixel is determined based on the fact that a slanted straight line is quantized into a set of either horizontal or vertical line segments separated by one-pixel steps.

In addition, we assume that the boundary noise is no more than one pixel. If this noise level is known *a priori* then this threshold can be adjusted accordingly [39].

Merging is done in the following manner. For each pair of adjacent line segments comprising three successive break-points p_l, p_{l+1} and p_{l+2} (say), if the absolute perpendicular distance of all points intermediate of p_l and p_{l+2} from the line segment $p_l p_{l+2}$ does not exceed one-pixel then the point p_{l+1} is merged, otherwise this point is retained.

2.2 Algorithm

The input are the data points $(x_i\ ,\ y_i)$, $i = 1, 2, ..., n$. The output are vertices p_l of the polygon.

Begin

Step 1. Initial segmentation

Compute

$$\theta_i = cos^{-1}\left(\frac{\mathbf{c}_i.\mathbf{c}_{i+1}}{|\,\mathbf{c}_i\,|\,|\,\mathbf{c}_{i+1}\,|}\right), \qquad i = 1, 2, ..., n$$

and

$$r_i = (4/\pi)\,\theta_i\,, \qquad i = 1, 2, ..., n\,.$$

If there exist at least two values of i for which $r_i \geq 3$, then these points constitute the initial set of breakpoints

else if there exists only one i for which $r_i \geq 3$ then this point together with those with $r_i = 2$ constitute the initial set of break-points

else if there exists no point with rank $r_i \geq 3$ and if there exist at least two points with $r_i = 2$ then these points constitute the initial set of break-points

else if there exists only one point with $r_i = 2$ then this point together with those with $r_i = 1$ comprise the initial set of break-points

else if there exists no point with $r_i = 2$ but there exist at least two points with $r_i = 1$ then these points comprise the initial set of break-points.

Step 2. Splitting

Compute

$$d_k = \frac{|(y_j - y_i)x_k - (x_j - x_i)y_k - x_i y_j + x_j y_i|}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}},$$

where $k = i+1, i+2, ..., j-1$.

If $d_k > 1$ then the segment $p_i p_j$ is split at the point where d_k is a maximum

else if $d_k \leq 1$ then no splitting is necessary.

After all necessary splittings are done go to Step 3.

Step 3. Merging

For every three successive break-points p_l, p_{l+1} and p_{l+2} the vertex p_{l+1} is merged with the segment $p_l p_{l+2}$ if the distance of every point intermediate of p_l and p_{l+2} from the segment $p_l p_{l+2}$ does not exceed 1

else merging is not possible.

After all necessary merging is done go to Step 4.

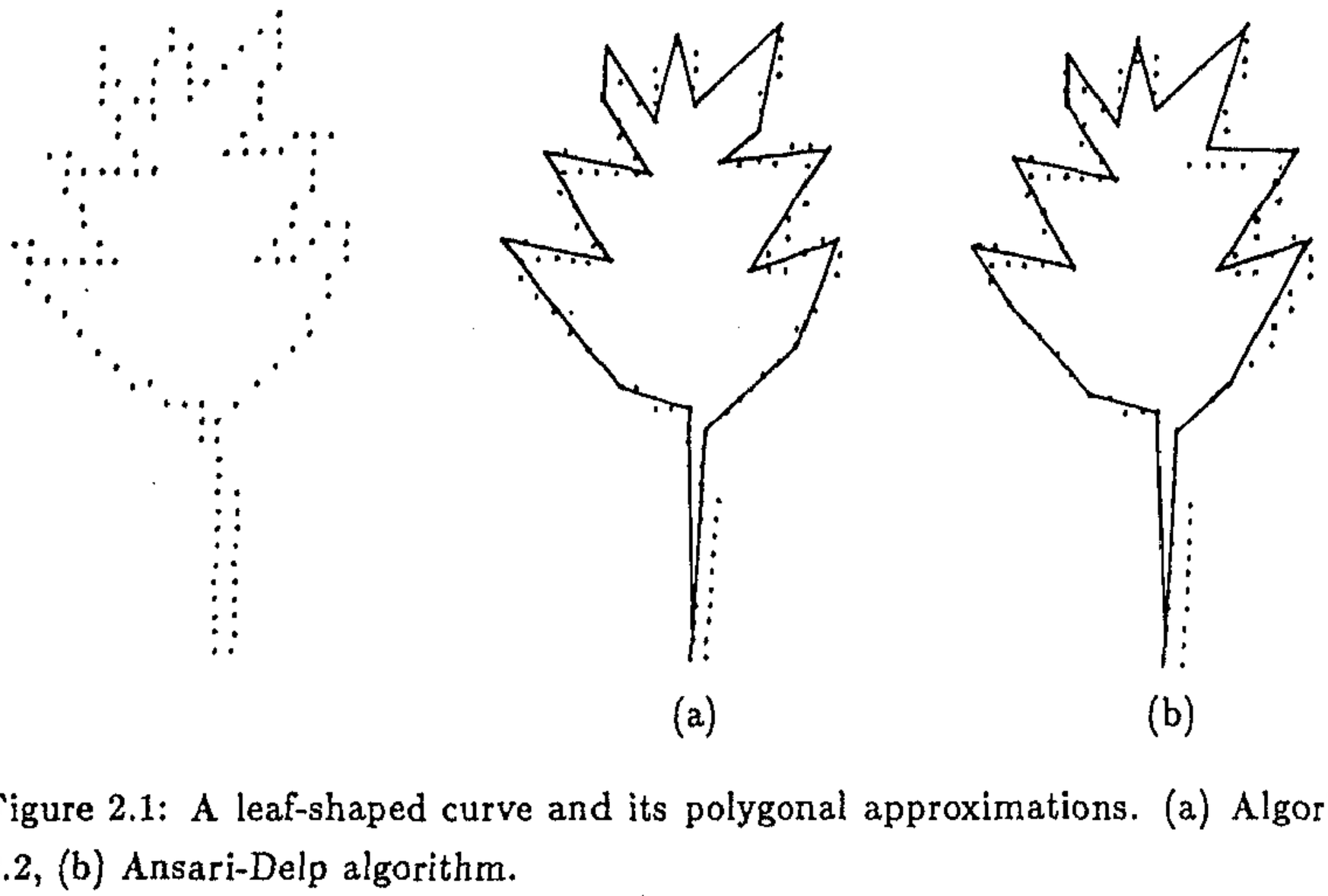
Step 4. Repeat Step 2 and Step 3 until an equilibrium is reached.

End.

2.3 Experimental results

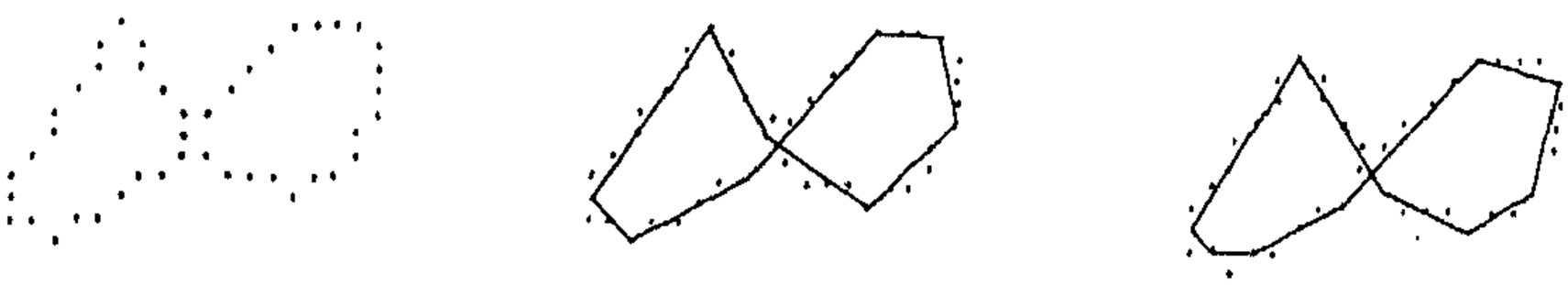
To focus on the performance of the algorithm 2.2 we apply it on four digital curves, namely, a leaf-shaped curve (Figure 2.1), a figure-8 curve (Figure 2.2), an aircraft (Figure 2.3) and a screwdriver (Figure 2.4). The first two of these are taken from Rosenfeld and Johnston [48], the third is taken from Gupta and Malakapalli [23] and the fourth is from Medioni and Yasumoto [28]. The polygonal approximations are shown in Figures 2.1 through 2.4. These figures also show the approximations as obtained by the Ansari-Delp algorithm [3].

As seen from these approximations the algorithm 2.2 places the vertices at the position where they should be (as judged by perception) whereas, the Ansari-Delp algorithm sometimes fails to do so, shifting the vertices from their actual position, rounding off sharp turnings (see Figure 2.3 and the upper right part of the Figure 2.1). An overview of the results of the approximations as obtained by algorithm 2.2 and those by the Ansari-Delp algorithm are displayed in table 2.1. The table shows the number of data points (n) of the digital curve, the number of vertices (n_v), the



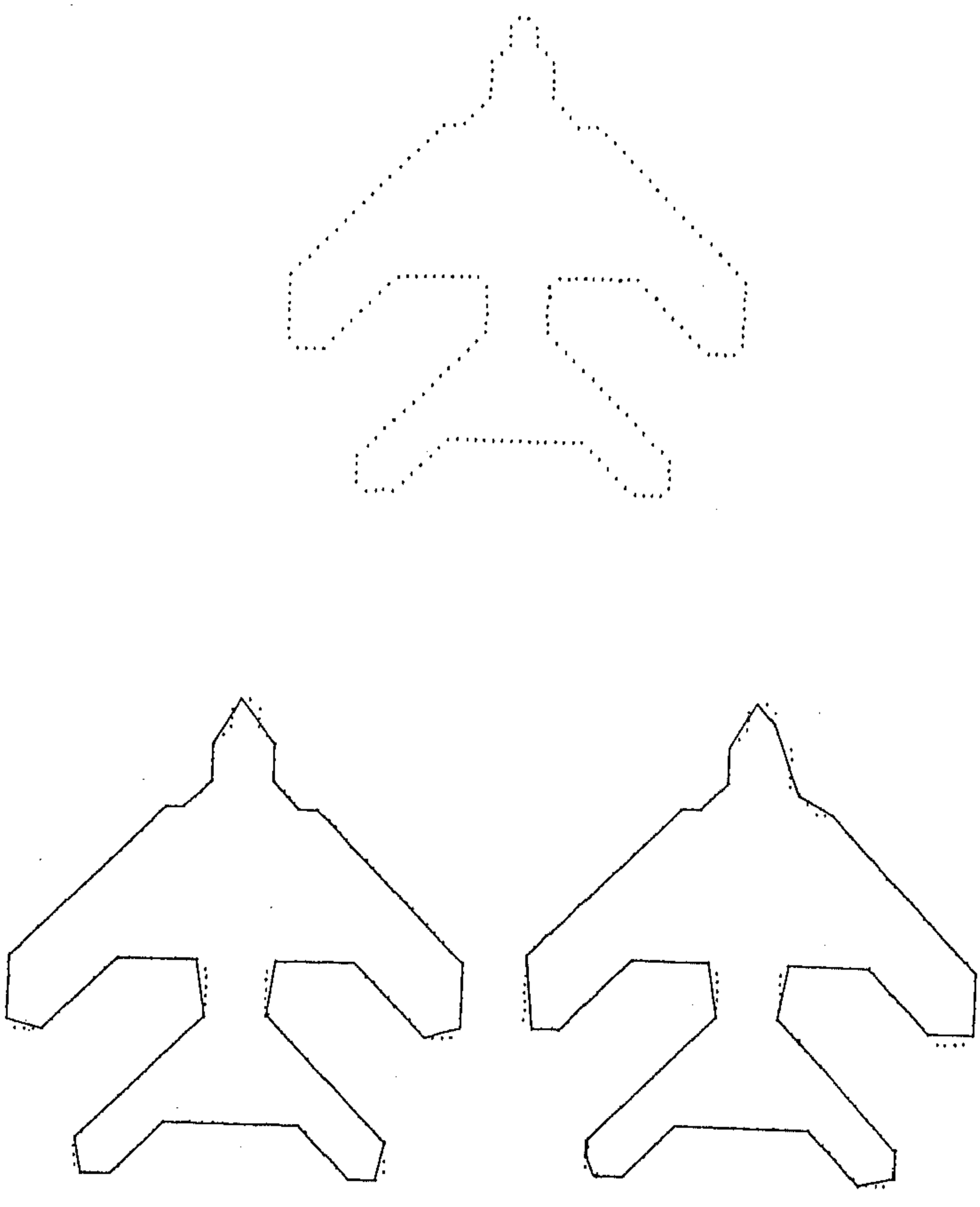
(a) (b)

Figure 2.1: A leaf-shaped curve and its polygonal approximations. (a) Algorithm 2.2, (b) Ansari-Delp algorithm.



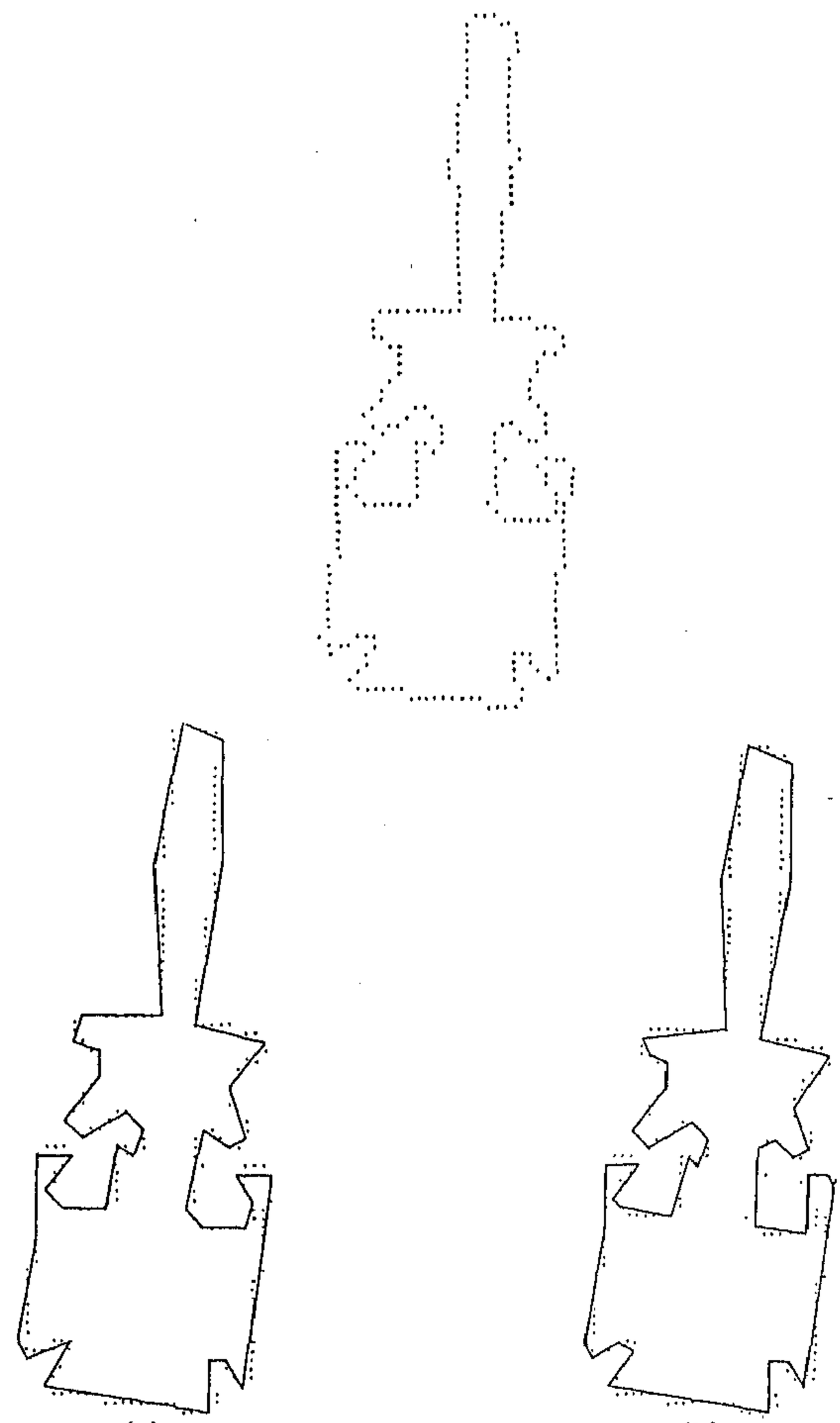
(a) (b)

Figure 2.2: A figure-8 curve and its polygonal approximations. (a) Algorithm 2.2, (b) Ansari-Delp algorithm.



(a) (b)

Figure 2.3: An aircraft and its polygonal approximations. (a) Algorithm 2.2, (b) Ansari-Delp algorithm.



(a) (b)

Figure 2.4: A screwdriver and its polygonal approximations. (a) Algorithm 2.2, (b) Ansari-Delp algorithm.

compression rate (n/n_v), percentage of data reduction, integral square error and the maximum allowable error (which is 1.00 in all cases). As seen from this table the algorithm 2.2 consistently outperforms the Ansari-Delp algorithm with respect to integral square error, the number of vertices and compression rate / percentage of data reduction for given maximum allowable error. For each of the digital curves the integral square error as obtained by the algorithm 2.2 is smaller than that obtained by the Ansari-Delp algorithm, though the number of vertices as obtained by algorithm 2.2 is not more than that obtained by the Ansari-Delp algorithm and consequently, the percentage of data reduction / compression rate as obtained by the algorithm 2.2 is no less than that obtained by the Ansari-Delp algorithm. Precisely speaking, the number of vertices of the figure-8 curve and of the screwdriver image as obtained by algorithm 2.2 is smaller than that obtained by the Ansari-Delp algorithm and as a consequence of it the percentage of data reduction / compression rate as obtained by the algorithm 2.2 is greater than that obtained by the Ansari-Delp algorithm but the integral square error as produced by algorithm 2.2 is less than that produced by the Ansari-Delp algorithm. As regard to the leaf-shaped curve and the aircraft image the number of vertices / compression rate / percentage of data reduction as obtained by either of the two algorithms are the same but the integral square error as obtained by algorithm 2.2 is lower than that obtained by the Ansari-Delp algorithm. This shows that the algorithm 2.2 produces integral square error which is no greater than that obtained by the Ansari-Delp algorithm without producing more vertices. Moreover, algorithm 2.2 does not shift vertices, it does not round off sharp turnings and produces symmetrical approximation from a symmetrical digital curve. Whereas, the Ansari-Delp algorithm does not have these merits. So we conclude that the algorithm 2.2 produces more accurate results than the Ansari-Delp algorithm without producing redundant vertices.

Table 2.1 A comparison between algorithm 2.2 & Ansari-Delp algorithm				
Digital curve	Leaf	Figure-8	Aircraft	Screwdriver
Number of points(n)	120	45	200	267
Results of algorithm 2.2				
Number of vertices(n_v)	20	9	29	41
Compression ratio(n/n_v)	6.00	5.00	6.90	6.51
Percentage of data reduction	83.33	80.00	85.50	84.60
Integral square error	21.24	5.27	8.71	55.75
Maximum error	1.00	1.00	1.00	1.00
Results of Ansari-Delp algorithm				
Maximum allowable error	1.00	1.00	1.00	1.00
Number of vertices(n_v)	20	10	29	43
Compression ratio(n/n_v)	6.00	4.50	6.90	6.21
Percentage of data reduction	83.33	77.80	85.50	83.90
Integral square error	28.59	5.50	14.56	62.50
Maximum error	1.00	1.00	1.00	1.00

Chapter 3

A sequential one-pass method

In this chapter we present a sequential one-pass technique for polygonal approximation of digital curves with uniformly spaced points. The procedure [40] is based on a result from numerical analysis and some concepts of regression analysis. Here we find the first order finite differences of the abscissa (x) and the ordinates (y) of the data points describing the digital curve. Then we try to identify some patterns (described later) in the first order finite differences Δx and Δy. By recognizing these patterns we find the sides of a polygon approximating a curve.

3.1 Pattern recognition of finite differences

From the boundary point data (x_i , y_i) we construct the first order finite differences Δx_i and Δy_i using

$$\Delta x_i = x_{i+1} - x_i \quad \textit{and} \quad \Delta y_i = y_{i+1} - y_i \,. \tag{3.1}$$

For digital curves with uniformly spaced point these differences take values 0, 1, -1 only. The successive differences, as we go through them exhibit one of the following patterns:

a) Δx_i's are constant for a series of equidistant values of y.

b) Δy_i's are constant for a series of equidistant values of x.

c) Δx_i's take values 0 and 1 only for a series of equidistant values of y.

d) Δx_i's take values 0 and -1 only for a series of equidistant values of y.

e) Δy_i's take values 0 and 1 only for a series of equidistant values of x.

f) Δy_i's take values 0 and -1 only for a series of equidistant values of x.

3.2 Basic results and concepts

The polygon representation of the shape of the boundary of an object can be obtained using any boundary tracking algorithm. It is not uncommon that such a representation involves a polygon having too many vertices. Since the time required for processing a polygon is dependent on the number of vertices, this polygon is usually determined using a data smoothing technique. Our polygonal approximation, too, involves a smoothing technique which is based on the following proposition from numerical analysis [51] and some concepts from regression analysis.

Proposition 3.1 If the nth differences of a tabulated function are constant when the values of the independent variable are taken in arithmetic progression then the function is a polynomial of degree n.

Besides the above proposition we use some concepts of regression analysis. In the regression analysis the scatter diagram of the bivariate data exhibits either a linear or a curvilinear tendency. Either the points of the scatter diagram exhibit that there is a tendency in the points to cluster around a straight line or they exhibit a tendency of clustering around a curve (curvilinear tendency), provided the variables are related. If the points of the scatter diagram exhibit a linear tendency then we can find a linear relationship between the variables i.e. the points can be approximated by a straight line. In regression analysis this line is the least squares line. In our smoothing technique we exploit the above concepts without using least squares line.

3.3 Smoothing technique

The smoothing technique involves reading the first order finite differences Δx_i and Δy_i, locating the breakpoints and joining the successive breakpoints by straight line segments. In this section we describe how to perform the smoothing so as to locate the breakpoints.

Following the proposition stated in the last section, the series of points where

pattern (a) is observed can be smoothed out by a single straight line segment which is obtained by joining the end points of the series. The series of points revealing pattern (b) can similarly be smoothed out following the same proposition.

The series of successive points revealing pattern (c) exhibit that there is a tendency in the points to cluster around a straight line. Hence from the concept of regression analysis stated in the last section, these points can be approximated by a straight line segment. The straight line segment that is used here is not the least squares line. The use of least squares line sometimes leads to unconnected boundary, since the estimated line does not necessarily have to go through any of the points of the series. The straight line segment that is used to smooth out these points is obtained by joining the endpoints of the series revealing the pattern (c). The series of points revealing pattern (d), (e) and (f) are similarly smoothed out by a straight line segment.

In order to identify pattern (a) or (b) a counter c is introduced that will count the number of finite differences of x (or y) that remain constant for a series of equidistant values of y (or x). We set the critical value of this counter to 4. Larger values of c will produce less number of vertices at the cost of higher approximation error (please see chapter 9). Thus if at least four successive differences of x (or y) remain constant for a series of equidistant values of y (or x) then the pattern (a) (or (b)) is identified. The approximation error is controlled by this counter. Setting this counter to different values different approximations can be obtained.

3.4 Complexity

From the above description of the smoothing technique it is clear that the procedure need no numerical computation except subtractions. For n boundary points $2n$ subtractions are required. The procedure mainly involves comparisons. One comparison is needed for each value of i and so a total of $2n$ comparisons are required. This shows that the complexity of the procedure is $O(n)$.

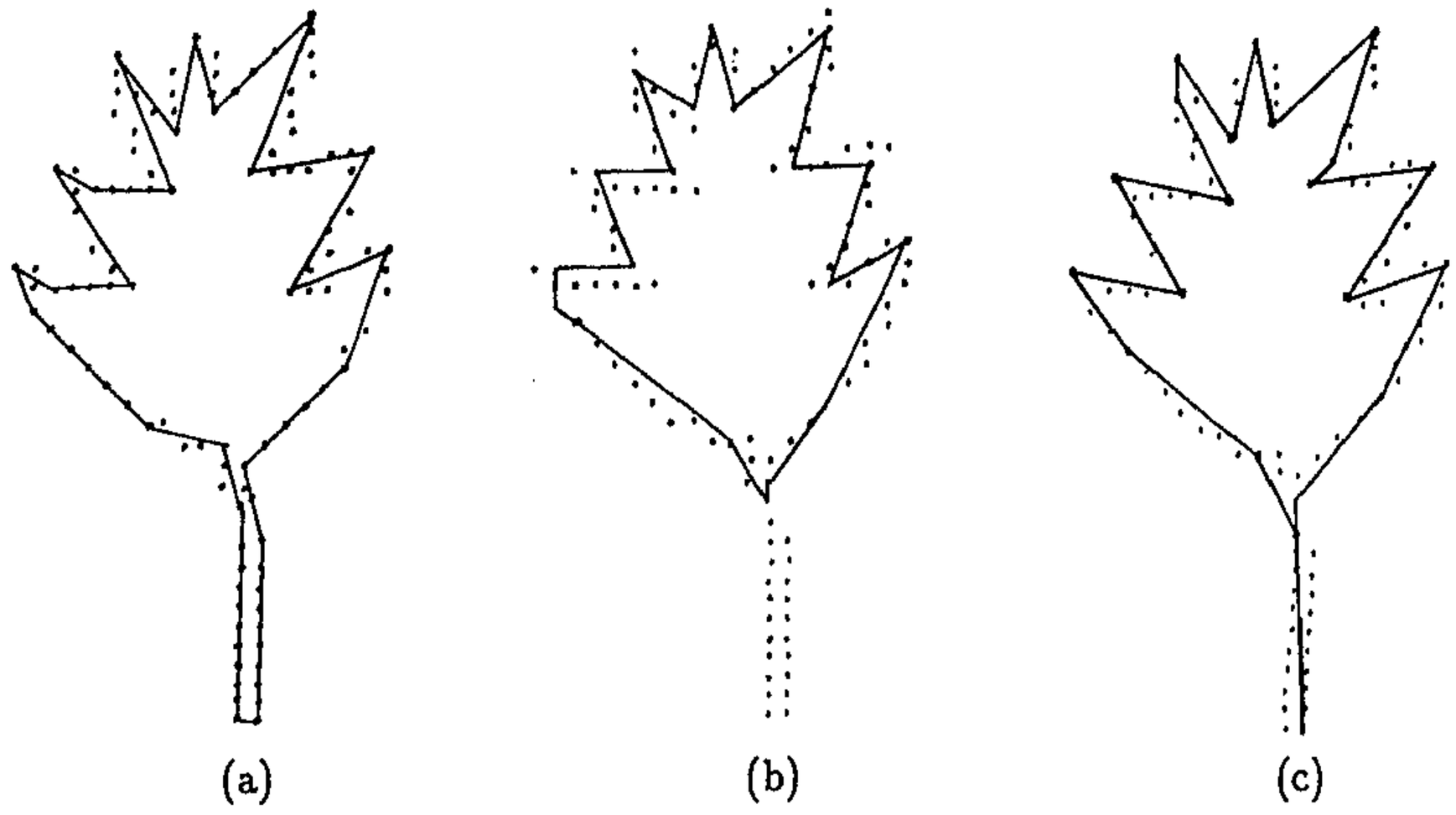


Figure 3.1: Polygonal approximations of the leaf-shaped curve. (a) Smoothing technique 3.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

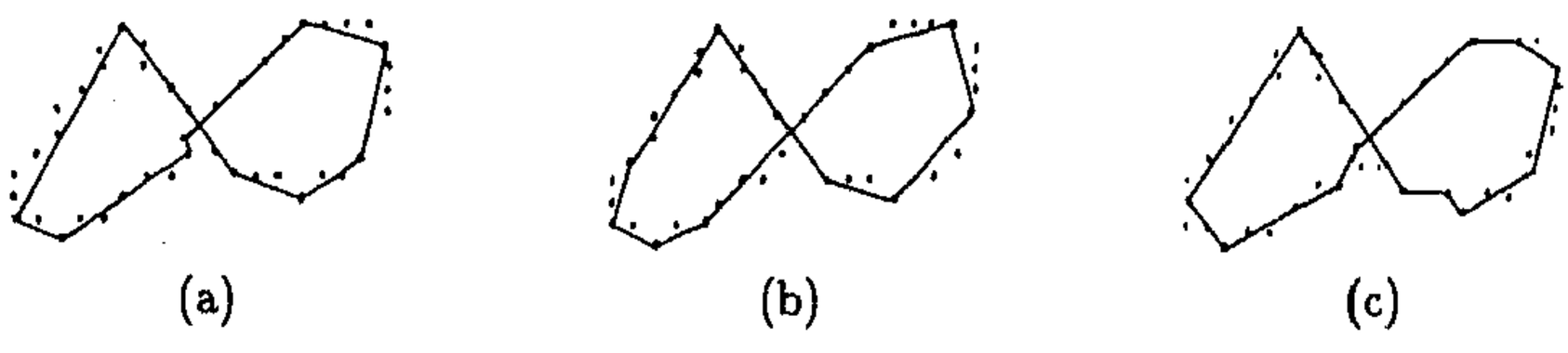
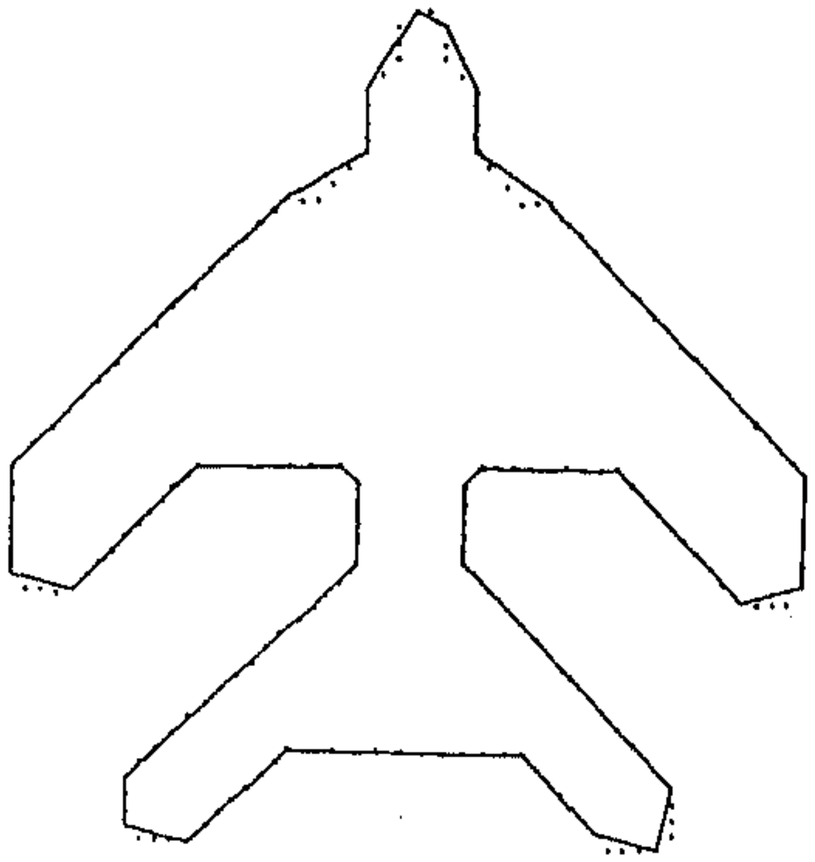
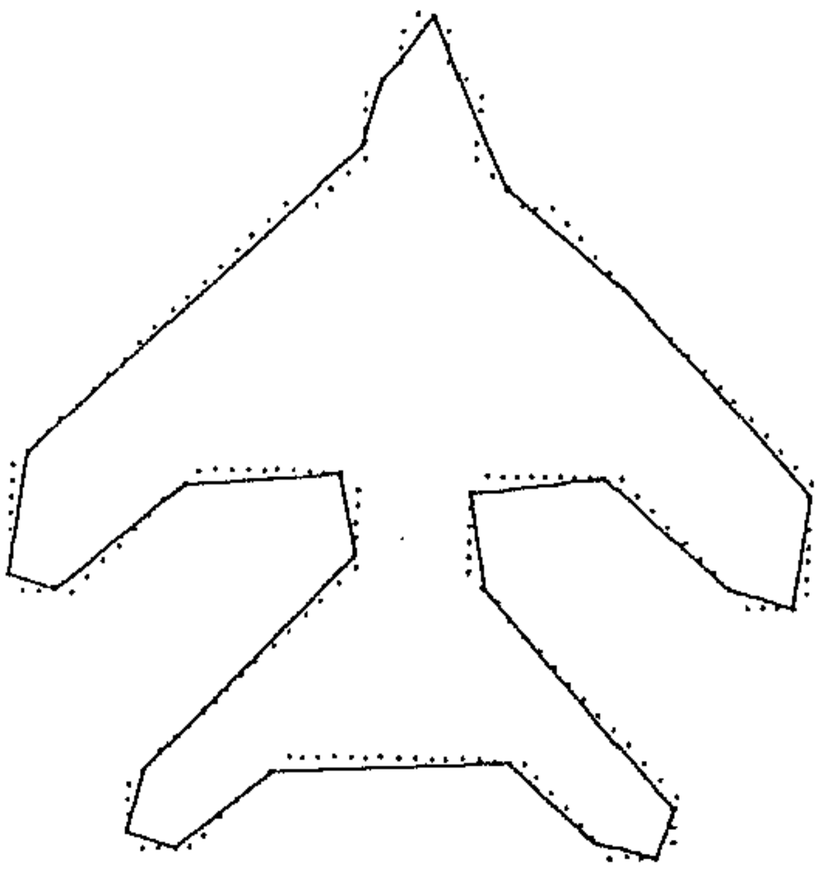


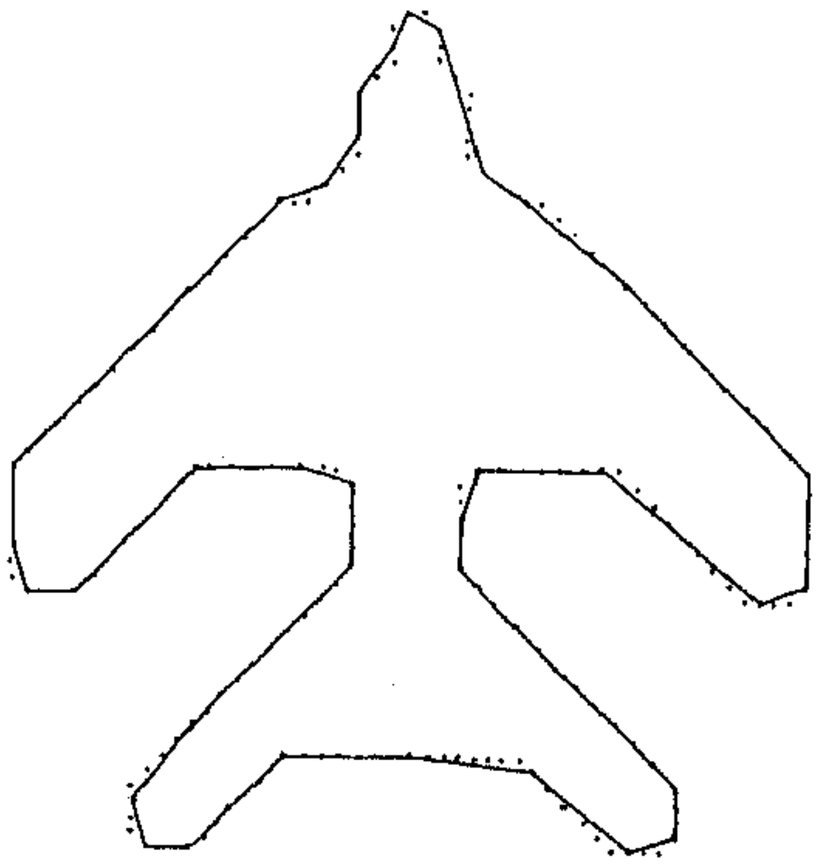
Figure 3.2: Polygonal approximations of the figure-8 curve. (a) Smoothing technique 3.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)

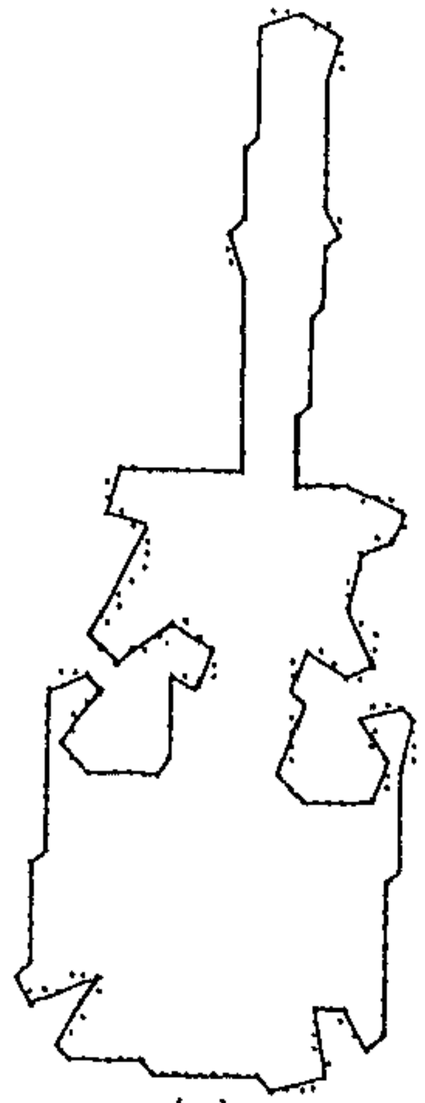


(b)

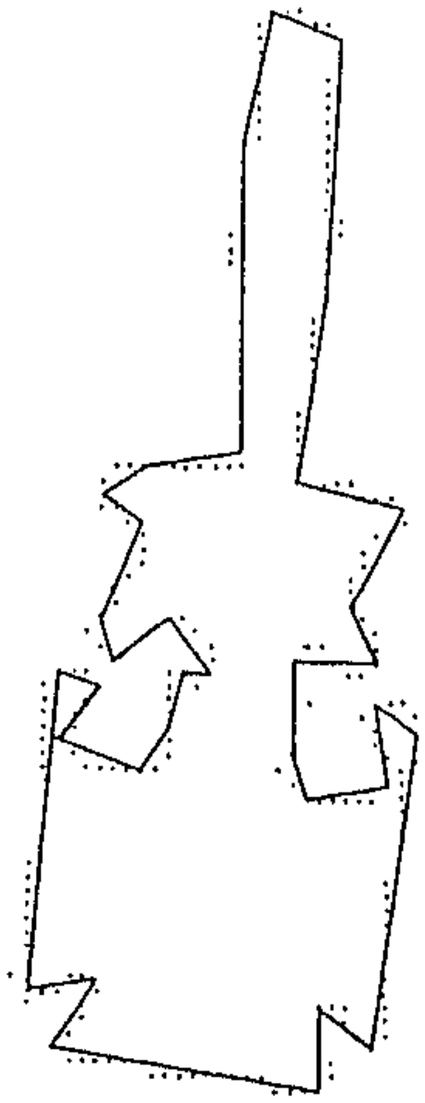


(c)

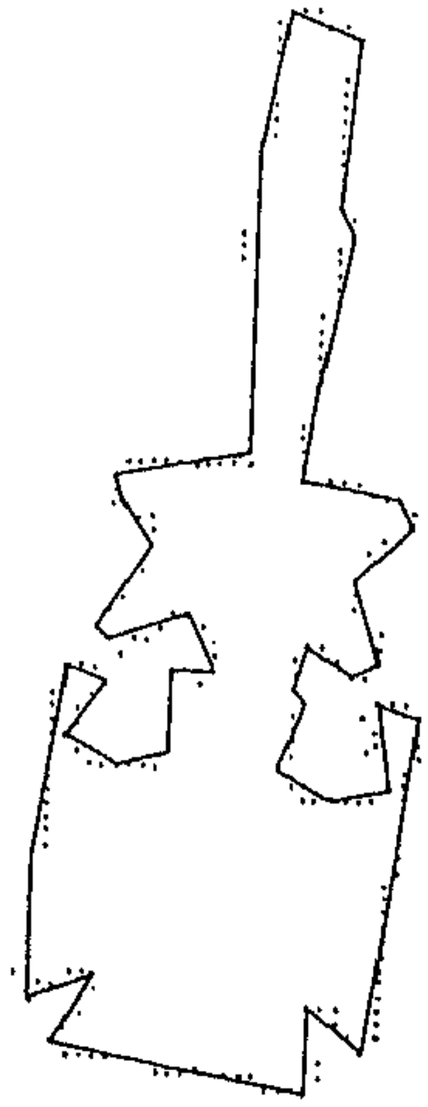
Figure 3.3: Polygonal approximations of the aircraft. (a) Smoothing technique 3.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)



(b)



(c)

Figure 3.4: Polygonal approximations of the screwdriver. (a) Smoothing technique 3.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

3.5 Experimental results

We have applied the smoothing technique 3.3 on four digital curves, namely a leaf-shaped curve (Figure 3.1), a figure-8 curve (Figure 3.2) an aircraft (Figure 3.3) and a screwdriver image (Figure 3.4) as in the last chapter. The polygonal approximations of the curves are shown in Figures 3.1 through 3.4. Since this algorithm is sequential we compare it with the Williams' algorithm [59] and the Wall-Danielsson algorithm [58] both of which are sequential in nature. As already stated the smoothing technique 3.3 involves a counter c and the maximum error is controlled by this counter. The maximum error as obtained by this algorithm is used to run Williams' algorithm on the same digital curves. The polygonal approximations are shown in Figures 3.1 through 3.4. As seen from these figures the worst approximation is obtained with the leaf-shaped curve (Figure 3.1) where the lower part of the curve which has a sharp turning is completely wiped out by the approximation and almost none of the other turnings are approximated according to their nature. The corners are always shifted to the nearby point. The vertices are located at a point away from where they should be. This is true for other digital curves also. But the approximations as obtained by the smoothing technique described in section 3.3 does not have any such deficiency. Neither does it round off sharp turnings nor does it miss corners. To compare our procedure with the Wall-Danielsson algorithm we use the maximum error returned by our procedure as the stopping criterion of the Wall-Danielsson algorithm. As already stated apart from the peak test the criterion function of the Wall-Danielsson algorithm is the area deviation per unit length. For comparison the threshold on the area deviation is initially set to a large value and is then diminished until the maximum error returned by our procedure is surpassed. The polygonal approximations as obtained by the Wall-Danielsson algorithm are shown in Figures 3.1 through 3.4. These approximations show that though the Wall-Danielsson algorithm retains the peaks but it fails to locate the vertices at their actual position when the turning is not so sharp (see Figure 3.3). An overview of the results of the approximations as obtained by smoothing technique 3.3 and those obtained by the Williams' and the Wall-Danielsson algorithm are displayed in Table 3.1. The table shows the number

of vertices (n_v), compression ratio, percentage of data reduction, maximum error and integral square error.

Table 3.1 A comparison among smoothing technique 3.3, Williams' & Wall-Danielsson algorithm				
Digital curve	Leaf	Figure-8	Aircraft	Screwdriver
Number of points(n)	120	45	200	267
Results of smoothing technique 3.3				
Number of vertices(n_v)	24	10	30	72
Compression ratio(n/n_v)	5.00	4.5	6.67	3.31
Percentage of data reduction	80.00	77.78	85.00	73.00
Integral square error	16.45	5.30	8.88	20.95
Maximum error	1.18	0.97	1.03	1.34
Results of Williams' algorithm				
Maximum allowable error	1.18	0.97	1.03	1.34
Number of vertices(n_v)	18	10	30	72
Compression ratio(n/n_v)	6.67	4.50	7.41	7.63
Percentage of data reduction	85.00	77.78	86.50	86.89
Maximum error	1.17	0.73	1.03	1.30
Integral square error	46.63	5.99	44.00	94.27
Results of Wall-Danielsson algorithm				
Maximum allowable error	1.18	0.97	1.03	1.34
Threshold	0.996	0.63	0.64	0.958
Number of vertices(n_v)	21	12	38	41
Compression ratio(n/n_v)	5.71	3.75	5.26	6.51
Percentage of data reduction	82.5	73.33	81.00	84.6
Maximum error	0.995	0.707	0.65	1.26
Integral square error	21.075	3.22	11.22	54.54

Chapter 4

Another sequential one-pass method

The procedure presented in the last chapter holds for digital curves with uniformly spaced points only. In this chapter we present another sequential one-pass algorithm [41] which holds for uniformly spaced points as well as non-uniformly spaced points. The procedure [41] is based on Pavlidis' [32] concept of almost collinearity of a sequence of points. Initially, the in-radius of triangles formed by the sequence of points triplets is introduced as a criterion function to measure the collinearity. This is an indirect approach but justified by a proposition which establishes that the higher the in-radius is, the higher is the perpendicular distance. Unfortunately, evaluation of in-radius is computationally expensive. To reduce the computational load the in-radius is replaced by the area and the perimeter of triangles. The vertices of the polygon are located by comparing the area and the perimeter with their critical values.

4.1 Approximation technique

If p_i , p_j and p_k , $(k = j+1)$ be three points of a digital curve C_d defined chapter 1, then the cross product of the vectors $p_i\vec{p}_j$ and $p_i\vec{p}_k$ is

$$p_i\vec{p}_j \times p_i\vec{p}_k = \{ (x_j - x_i)(y_k - y_i) - (y_j - y_i)(x_k - x_i) \}\,\hat{t} \qquad (4.1)$$

where $\hat{t}$ is a unit vector perpendicular to the plane determined by $p_i\vec{p}_j$ and $p_i\vec{p}_k$. The magnitude of the cross product is

$$| p_i\vec{p}_j \times p_i\vec{p}_k | = | (x_j - x_i)(y_k - y_i) - (y_j - y_i)(x_k - x_i) | . \tag{4.2}$$

Translating the origin of the coordinate system to the point p_i and denoting the new coordinate system by prime, the magnitude of the cross product reduces to the form

$$| \vec{op'_j} \times \vec{op'_k} | = | x'_j y'_k - x'_k y'_j | . \tag{4.3}$$

Again, the cross product of the vector $p_i\vec{p}_j$ and $p_j\vec{p}_k$ is

$$p_i\vec{p}_j \times p_j\vec{p}_k = \{ (x_j - x_i)(y_k - y_j) - (y_j - y_i)(x_k - x_j) \} \hat{t} \tag{4.4}$$

and its magnitude is

$$| p_i\vec{p}_j \times p_j\vec{p}_k | = | (x_j - x_i)(y_k - y_j) - (y_j - y_i)(x_k - x_j) | . \tag{4.5}$$

In prime coordinate system (4.5) reduces to

$$| \vec{op'_j} \times \vec{p'_j p'_k} | = | x'_j(y'_k - y'_j) - y'_j(x'_k - x'_j) | . \tag{4.6}$$

Both the magnitude (4.3) and (4.6) are twice the area (A) of the triangle with vertices at p_i, p_j, p_k that is,

$$2A = | x'_j y'_k - y'_j x'_k | , \tag{4.7}$$

$$2A = | x'_j(y'_k - y'_j) - y'_j(x'_k - x'_j) | . \tag{4.8}$$

The last form (4.8) is useful for chain-coded curves where $y'_k - y'_j$ and $x'_k - x'_j$ take values 0, 1 and -1 only.

Proposition 4.1. If the area of a triangle with p_i, p_j and p_k as vertices be zero / almost zero (i.e. within a specified value) then the points p_i, p_j and p_k are collinear / almost collinear.

Proposition 4.2. If three points p_i, p_{i+1} and p_{i+2} be collinear / almost collinear and the three points p_i, p_{i+2} and p_{i+3} be also collinear / almost collinear then the points p_i, p_{i+1}, p_{i+2} and p_{i+3} are collinear / almost collinear. Again, if p_i, p_{i+3} and

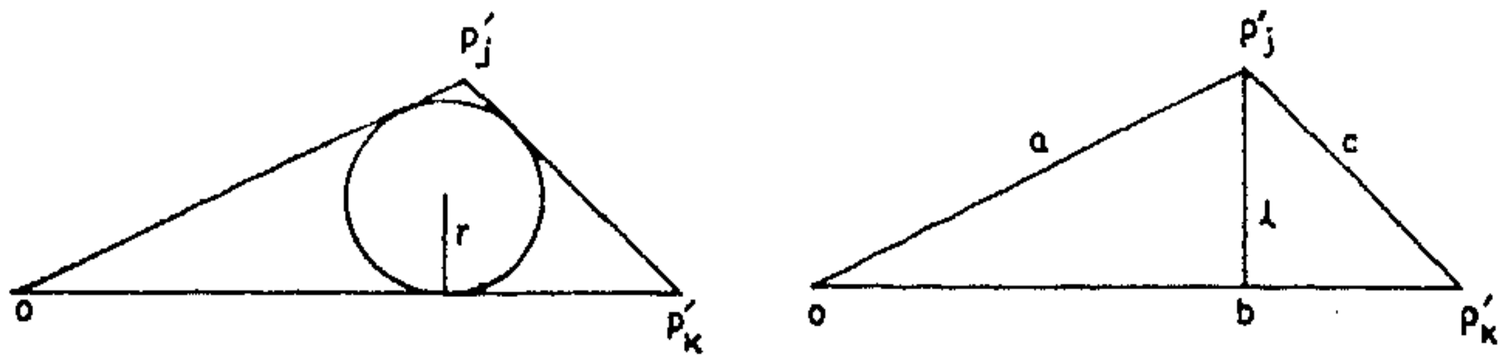


Figure 4.1: The in-radius and perpendicular distance

p_{i+4} be collinear / almost collinear then the points p_i , p_{i+1} and p_{i+2} , p_{i+3} and p_{i+4} are collinear / almost collinear. And this process can be carried on for any finite number of points.

Using the Propositions 4.1 and 4.2 and taking the measure (4.7) for curves with non-uniformly spaced points and (4.8) for curves with uniformly spaced points (chain-coded curve) we can check the collinearity of a sequence of points by specifying the critical value of $2A$. But the choice of this critical value is problematic. If we set it to 1, we get many redundant vertices and if we set it to 2 we miss many important vertices and so we lose the shape of the curve. So it is not effective to use $2A$ only as a measure of collinearity. In the following we introduce

$$r = \frac{A}{s} = \frac{2A}{2s} = \frac{magnitude\ of\ cross\ product}{perimeter\ of\ triangle\ op'_jp'_k} \tag{4.9}$$

as an alternative measure of collinearity. The geometrical significance of r is that it gives the in-radius of the triangle $p_ip_jp_k$ $(op'_jp'_k)$. This is an indirect approach to the problem but is justified by the following proposition.

Proposition 4.3. In any triangle the in-radius (r) and the absolute perpendicular distance (l) of a vertex from the opposite side are related by $r < \frac{1}{2}l$.

Proof. Since $r = \frac{A}{s}$, referring to Figure 4.1 $r = \frac{b.l}{a+b+c}$. In any triangle, $c + a > b$, so, $\frac{c+a}{b} + 1 > 2$ i.e. $\frac{a+b+c}{b} > 2$ i.e. $\frac{b.l}{a+b+c} < \frac{1}{2}l$ i.e. $r < \frac{1}{2}l$.

This result shows that the higher the in-radius is, the farther is the point p'_j (p_j) from the line op'_j (p_ip_j). So we may take r as a measure of prominence for the point p'_j to be a vertex. We can decide upon a critical value of r and comparing r with its critical value we can locate the vertices. This procedure fails to catch the peaks of a curve. Sometimes peaks also play an important role to detect defects of an object

[5]. Observation shows that as the procedure passes a peak the in-radius maintains non-zero constant value which is below its critical value. So it is possible to locate the vertices accurately if we incorporate a check on two successive non-zero values of r, in addition to comparing it with its critical value.

Unfortunately, the evaluation of r is computationally expensive, because evaluation of perimeter need evaluation of three square roots (the length of three vectors $\vec{op'_j}$, $\vec{op'_k}$ and $\vec{p'_jp'_k}$. So we approximate these lengths by the maximum of the absolute value of the components of the vector. Thus the length of $\vec{op'_j}$, $\vec{op'_k}$ and $\vec{p'_jp'_k}$ are approximated by

$$\begin{aligned} |\vec{op'_j}| &= max(|x'_j|, |y'_j|), \\ |\vec{op'_k}| &= max(|x'_k|, |y'_k|), \\ |\vec{p'_jp'_k}| &= max(|x'_k - x'_j|, |y'_k - y'_j|). \end{aligned} \tag{4.10}$$

Using this approximation we may use r to locate the vertices by the procedure described earlier. But the evaluation of r involves division of the magnitude of the cross product by the perimeter of the triangle. We avoid this arithmetic operation by dispensing with the measure r and replacing it by two measures namely, $2A$ and $2s$. Now we can compare these two measures with their critical values 1 and 3 respectively to check the collinearity of a sequence of points.

Since the comparison of r with its critical value fails to catch the peaks hence comparison of $2A$ and $2s$ with their critical value will also fail to do the same. But if we use r as a measure of collinearity, to catch the peaks we incorporate a check on two successive non-zero values of r when it is below its critical value. Similarly if we use $2A$ and $2s$ as a measure of collinearity we incorporate a check on two successive values of $2s$ when it is above its critical value and $2A$ is equal to 1.

So instead of using r as a measure of collinearity we use $2A$ and $2s$ as a measure of collinearity. This replacement of r by $2A$ and $2s$ reduces the computational load significantly and at the same time they perform the same task and produce the same results as obtained using r only. Further using two measures we can make a good compromise between unwanted vertices and loss of important vertices which occur when we consider only one measure $2A$.

The collinearity check can be started from an arbitrary point and is carried

beyond the starting point till the vertex generated last coincides with one of the vertices already generated.

With this discussion we are now ready to present the algorithm for detecting the vertices of the polygon. We propose to denote the magnitude of the cross product (4.7) and (4.8) and the approximate value of the perimeter of the triangle formed by the points p_i , p_j and p_k $(k = j + 1)$ by D_k and P_k respectively.

4.2 Algorithm

Comments: The input are the coordinates (x_i , y_i), $i = 1, 2, ..., n$. The output are the vertices. All arithmetics are performed in integer mode. All arithmetics are in modulo n.

Begin

Step 1. Initiate $i = 1$.

Step 2. Translate the coordinate system to the point p_i so that it becomes the origin of the prime coordinate system; set $j = i + 1$; $k = i + 2$.

Step 3. Compute

$$D_k = |x'_j(y'_k - y'_j) - y'_j(x'_k - x'_j)|, \quad \textit{for chain coded curve}$$
$$D_k = |x'_j y'_k - y'_j x'_k|, \quad \textit{elsewhere.}$$

Step 4. If $D_k = 0$ then the point p'_k (p_k) passes the test; change j to $j + 1$; k to $k + 1$; go to Step 3

else if $D_k = 1$ then compute P_k by

$$P_k = max(|x'_j|, |y'_j|) + max(|x'_k|, |y'_k|) + max(|x'_k - x'_j|, |y'_k - y'_j|).$$

Step 5. If $P_k > 3$ and $P_k \neq P_{k-1}$ then p'_k (p_k) passes the test; change j to $j + 1$; k to $k + 1$; go to Step 3

else p'_j (p_j) is a vertex; write j; (x_j , y_j); go to Step 2.

Step 6. Repeat this process till the vertex generated last coincides with one of the vertices already generated.

Step 7. Join the vertices in order to obtain the polygon.

End.

4.3 Computational complexity

As it is clear we have used two forms of the same measure D_k, namely,

$$D_k = | x'_j(y'_k - y'_j) - y'_j(x'_k - x'_j) |$$

and

$$D_k = | x'_j y'_k - y'_j x'_k | .$$

The second form is deducible from the first form. We use the first form for chain coded curves (curves with uniformly spaced points) where $y'_k - y'_j$ and $x'_k - x'_j$ are either 0 or 1 or -1 and so the evaluation of D_k need no multiplication. For other curves we use the second form which involves two multiplications. Here if we use the first form instead of the second it will increase the total number of arithmetic operations (introducing two unnecessary subtractions). The evaluation of P_k involves no multiplication. This shows that our procedure need only two multiplications for digital curves with non-uniformly spaced points and no multiplication for chain coded curve.

4.4 Discussion

The polygonal approximation proposed by Williams [59] used scan along technique which need a single pass through data. Wall and Danielsson [58] too, used scan along technique but it is faster than that of Williams. For curves with uniformly spaced points Williams' technique need seven multiplication / divisions and evaluation of a square root. Whereas, Wall and Danielsson's technique need six multiplications and the simplified version of it requires only three to four multiplications. For curves with non-uniformly spaced points (chain coded curves) Williams' technique requires five multiplications whereas Wall and Danielsson's need two multiplications and the simplified version of it need only one multiplication. The algorithm 4.2 presented in this chapter requires only two multiplications for curves with non-uniformly spaced points and no multiplication for chain coded curves.

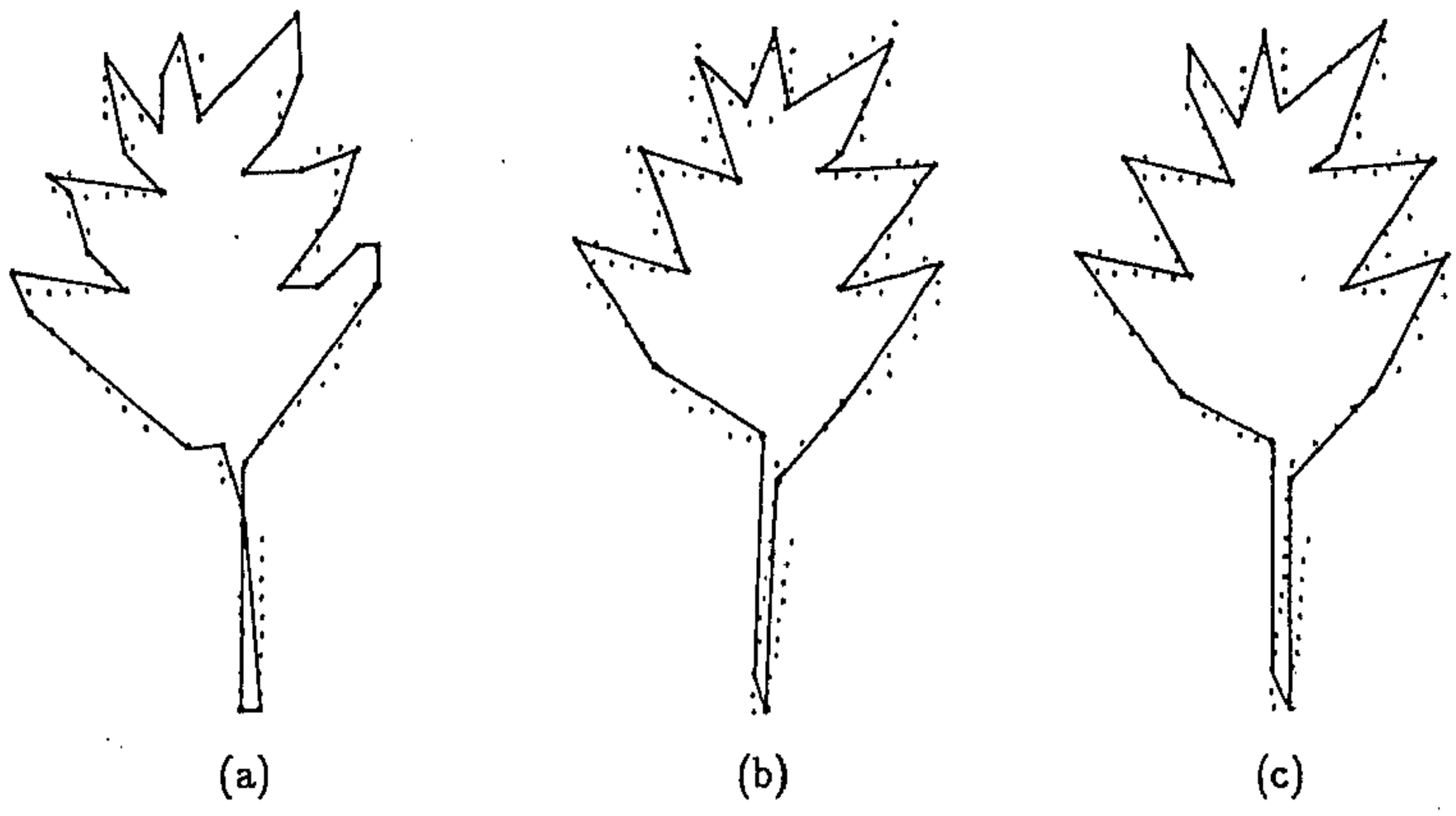
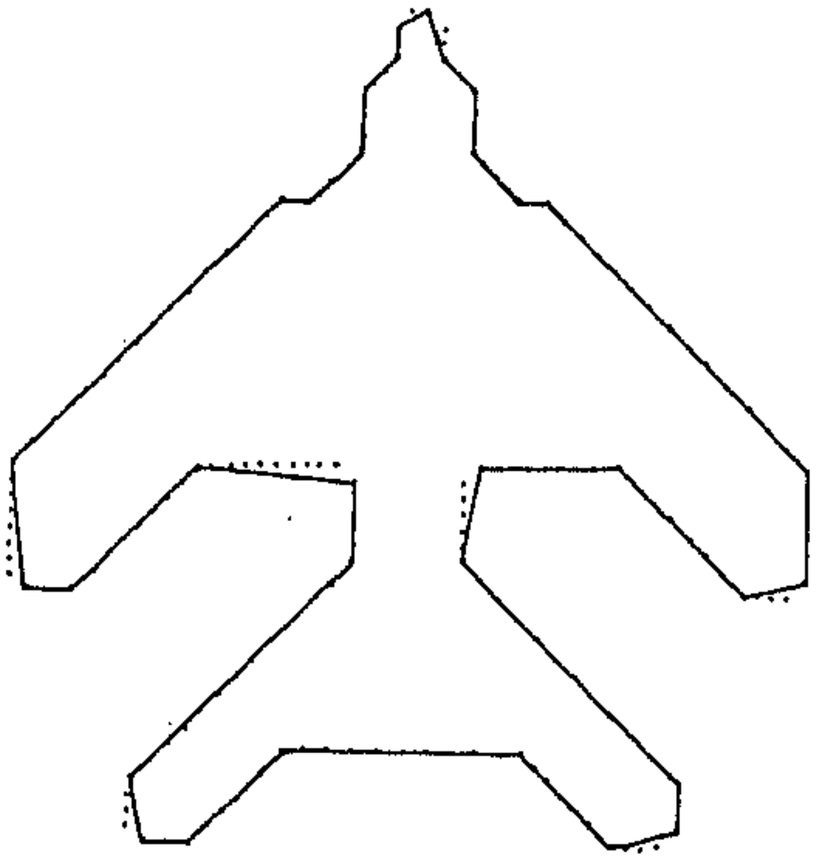
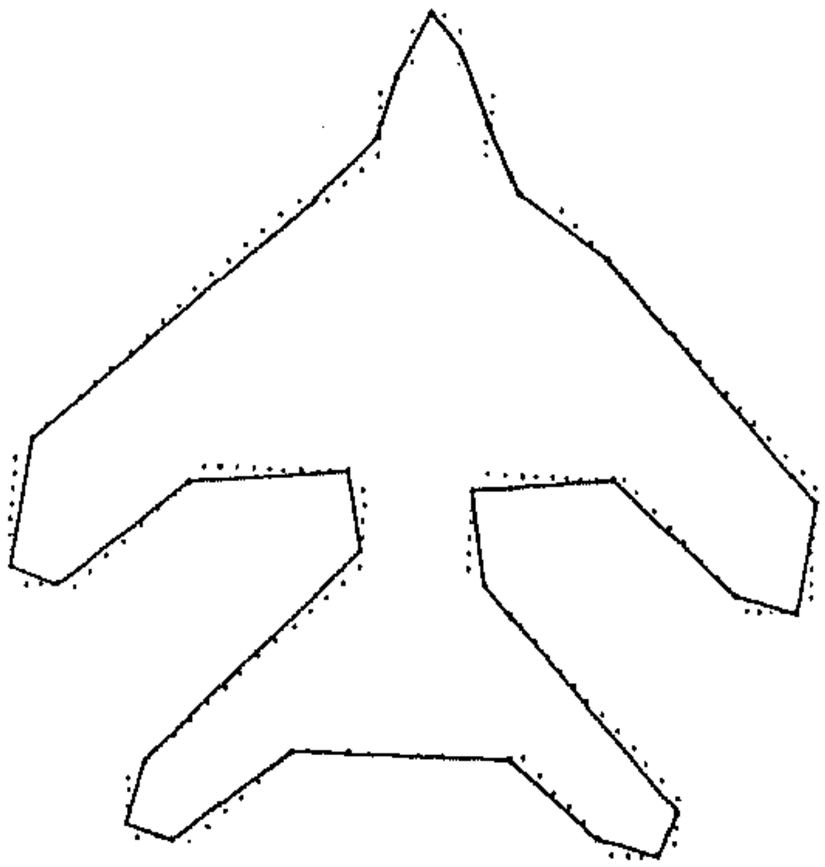


Figure 4.2: Polygonal approximations of the leaf-shaped curve. (a) Algorithm 4.2, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

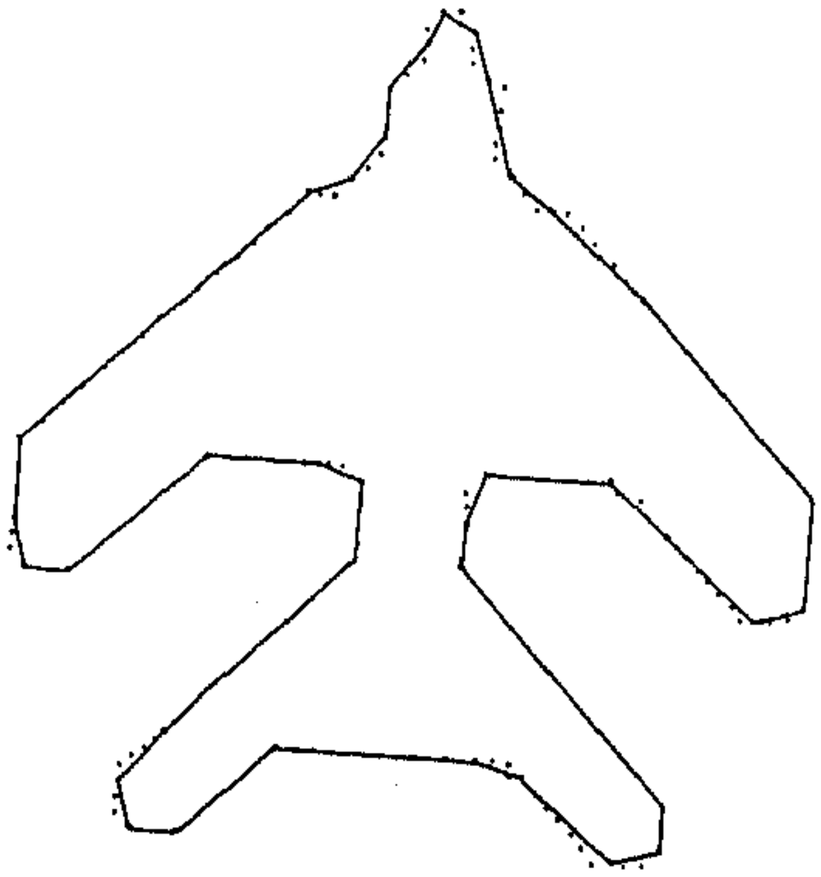
Figure 4.3: Polygonal approximations of the figure-8 curve. (a) Algorithm 4.2, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)

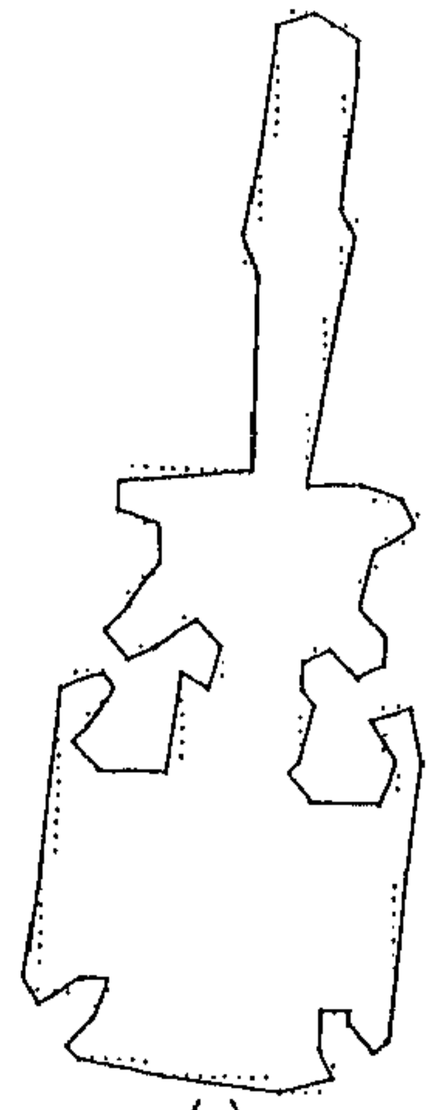


(b)

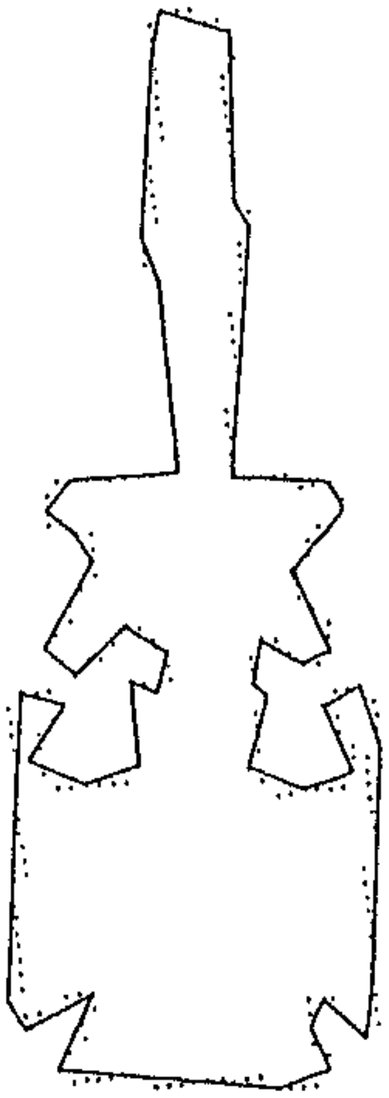


(c)

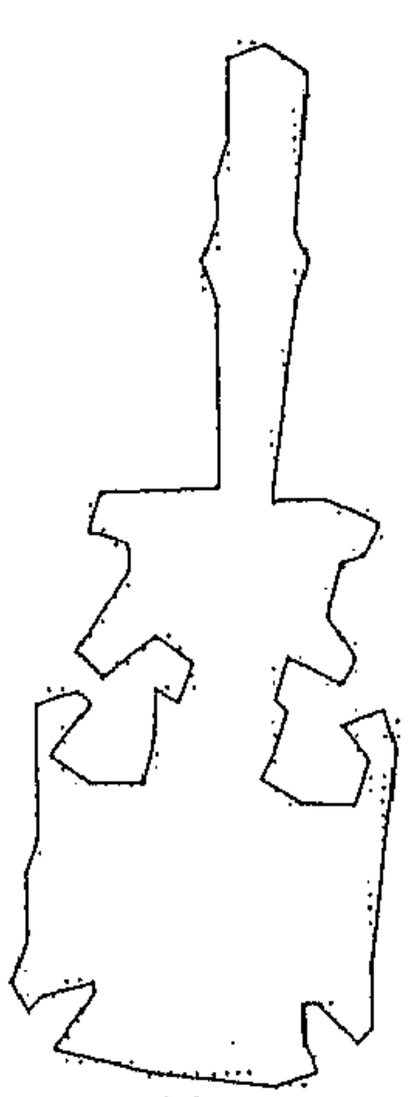
Figure 4.4: Polygonal approximations of the aircraft. (a) Algorithm 4.2, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)



(b)



(c)

Figure 4.5: Polygonal approximations of the screwdriver. (a) Algorithm 4.2, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

4.5 Experimental results

The algorithm 4.2 developed in this chapter is applied on the same digital curves as in the previous chapters. The digital curves and their polygonal approximations are shown in Figures 4.2 through 4.5. The data are processed in the clockwise direction. Since this procedure is sequential and one-pass hence we compare the experimental results of this procedure with those of the Williams' algorithm [59] and the Wall-Danielsson algorithm [58]. As we have already seen this procedure controls the maximum error indirectly, so the maximum error that is obtained by applying this procedure on each of the four digital curves are used to run the Williams' algorithm and the Wall-Danielsson algorithm on the same digital curves. The polygonal approximations as obtained by the Williams' algorithm and the Wall-Danielsson algorithm are shown in Figures 4.2 through 4.5. As seen from these figures the approximations as obtained by our procedure is better than those produced by the Williams' or the Wall-Danielsson algorithm. The Williams' algorithm misses corners and rounds off sharp turnings (please see the leaf and the aircraft). It also detects false vertices (please see the screwdriver). The worst approximation is obtained with the aircraft image (Figure 4.4) where most of the turnings are not approximated according to their nature. The vertices are shifted to the nearby point from their actual position. Though the Wall-Danielsson algorithm retains the peaks but it misses corners (please see the bottom of the leaf) and sometimes it detects false vertices (please see the aircraft and the screwdriver). A comparison of the results obtained by our procedure with those of the Williams' and Wall-Danielsson are shown in Table 4.1.

Table 4.1 A comparison among algorithm 4.2, Williams' & Wall-Danielsson algorithm				
Digital curve	Leaf	Figure-8	Aircraft	Screwdriver
Number of points(n)	120	45	200	267
Results of algorithm 4.2				
Number of vertices(n_v)	33	16	32	70
Compression ratio(n/n_v)	3.64	2.81	6.25	3.81
Percentage of data reduction	72.50	64.40	84.00	73.80
Integral square error	11.17	3.75	9.63	41.40
Maximum error	0.896	0.73	0.896	0.92
Results of Williams' algorithm				
Maximum allowable error	0.896	0.73	0.896	0.92
Number of vertices(n_v)	20	9	28	50
Compression ratio(n/n_v)	6.00	5.00	7.14	5.34
Percentage of data reduction	83.33	80.00	86.0	81.3
Maximum error	0.894	0.73	0.896	0.91
Integral square error	26.61	5.99	37.50	47.85
Results of Wall-Danielsson algorithm				
Maximum allowable error	0.896	0.73	0.896	0.92
Threshold	0.95	0.63	0.65	0.67
Number of vertices(n_v)	21	12	38	66
Compression ratio(n/n_v)	5.71	3.75	5.26	4.05
Percentage of data reduction	82.5	73.33	81.00	75.33
Maximum error	0.894	0.707	0.64	0.77
Integral square error	18.44	3.22	11.22	25.18

Chapter 5

A data-driven method

5.1 Departure from conventional approach

In the polygonal approximation techniques discussed so far the maximum allowable error is specified either directly or indirectly. The optimal algorithms for polygonal approximation which use the least squares principle ([55], [9], [19], [20], [21] and [34]) need the number of line segments to be specified. The optimal algorithm given by Dunham [14] need error specification. In all these procedures the user of the procedure has to specify either the number of line segments or the maximum allowable error. The maximum allowable error or the number of line segments to be used is determined on the basis of a trial and error process. So these procedures cannot run without operator's intervention.

In this chapter we are looking for a data-driven method [42] in which neither do we specify the error nor do we specify the number of line segments. We keep both these parameters free and allow the procedure to determine the length of the line segment as well as the maximum allowable error adaptively on the basis of the local topography of the curve. So the procedure does not need operator's intervention. Though the procedure is sequential and one pass, but unlike the existing sequential algorithms neither does it miss corners nor does it round off sharp turnings.

5.2 Procedure

The equation of a straight line joining the point p_i to the point p_j is

$$(y_j - y_i)x - (x_j - x_i)y - x_i y_j + x_j y_i = 0\,. \tag{5.1}$$

Translating the origin of the coordinate system to the point $(x_i\,,\,y_i)$ and denoting the new coordinate system by prime so that

$$x'_j = x_j - x_i \quad \textit{and} \quad y'_j = y_j - y_i \tag{5.2}$$

the equation (5.1) takes the form

$$y'_j x' - x'_j y' = 0\,. \tag{5.3}$$

The error e_k between the point $p_k\,,\,k = i+1,\,i+2,...,\,j-1$ and the line (5.3) is the perpendicular distance of the point p_k from the line (5.3) that is,

$$e_k = \frac{y'_j x'_k - x'_j y'_k}{\sqrt{{x'_j}^2 + {y'_j}^2}}\,. \tag{5.4}$$

So while approximating the points $p_k\,,\,k = i,\,i+1,\,i+2,...,\,j$; the integral square error along the line segment (5.3) is

$$s_j = \sum_{k=i+1}^{j-1} \frac{(y'_j x'_k - x'_j y'_k)^2}{{x'_j}^2 + {y'_j}^2}\,. \tag{5.5}$$

And the length of the line segment joining $(x_i\,,\,y_i)$ to $(x_j\,,\,y_j)$ is

$$l_j = \sqrt{{x'_j}^2 + {y'_j}^2}\,. \tag{5.6}$$

Our objective is to make l_j as large as possible by continuously merging points one after another with the point $(x_i\,,\,y_i)$ so that s_j is as small as possible. We note that we cannot increase l_j indefinitely nor can we decrease the error s_j arbitrarily. Because in the former case, the approximation error will be too large resulting in an approximation that may fail to locate many significant vertices. And in the later case, the approximation may result in many redundant vertices. So it is necessary to make a compromise between the length of the line segment and the

approximation errors. We solve this problem by combining l_j with s_j. Since our objective is to make l_j as large as possible so that s_j is as small as possible, we take the mathematical combination

$$f_j = l_j - s_j . \tag{5.7}$$

The procedure looks for the local maxima of f_j by continuously merging points one after with the point p_i. The value of j for which f_j attains a local maximum gives the location of a vertex.

The procedure is started from an arbitrary point p_i: The origin is shifted to the point p_i by the transformation rules (5.2), write $j = i+1$ and compute f_j using the formula

$$f_j = \sqrt{x_j'^2 + y_j'^2} \tag{5.8}$$

because for $j = i + 1$, $s_j = 0$. Then j is changed to $j + 1$ and f_j is computed using the formula (5.7) and f_j is compared with f_{j-1}. Points are merged one after another with the point p_i giving increment to j, f_j is computed using (5.7) and compared with f_{j-1} until f_j falls below f_{j-1}. At this stage $jj = j - 1$ gives the location of a vertex which is not necessarily a valid vertex. We write $i = jj$, translate the origin to the point p_i, set $j = i+1$ and compute f_j using (5.8). j is changed to $j + 1$ and f_j is computed using (5.7) and compared with f_{j-1}. Points are merged one after with the point p_i giving increment to j, f_j is computed using (5.7) and compared with f_{j-1} until f_j falls below f_{j-1}. When f_j falls below f_{j-1} another possible vertex is recorded at the point $jj = j - 1$. The coordinates of this vertex are (x_{jj}, y_{jj}). The origin is again translated to the point jj and the same computation is carried out in the same fashion as already described. The procedure is carried on along the entire digital curve beyond the starting point till the vertex generated last coincides with one of the vertices already generated.

At this point we note that Williams [59] and Wall and Danielsson [58], too looked for the maximal possible line segments. But in their procedure it is necessary to specify the error. In Williams' technique the error is specified directly whereas, in Wall and Danielsson's technique the error is specified indirectly in terms of area deviation per unit length. But the present procedure looks for the maximal line segments without specifying the error. We keep the error unspecified and allow the

procedure to determine it on the basis of the local topography of the curve.

5.3 Algorithm

The input are the data points $(x_i\,,\,y_i)$, $i = 1, 2, ..., n$. The output are the vertices $(x_{jj}\,,\,y_{jj})$ and jj. All arithmetics are in modulo n.

Begin

Step 1. Initiate $i = 1$.

Step 2. Translate the origin of the coordinate system to the point $(x_i\,,\,y_i)$ by the transformation rules

$$x'_j = x_j - x_i \quad and \quad y'_j = y_j - y_i\,.$$

Step 3. Set $j = i + 1$.

Step 4. Compute $f_j = \sqrt{x'^2_j + y'^2_j}$.

Step 5. Change j to $j + 1$.

Step 6. Compute $f_j = l_j - s_j$.

Step 7. If $f_j \geq f_{j-1}$ then go to Step 5

else write $jj = j - 1$ and $(x_{jj}\,,\,y_{jj})$ using

$$x_{jj} = x'_{jj} + x_i \quad and \quad y_{jj} = y'_{jj} + y_i\,;$$

set $i = jj$; go to Step 2.

Step 8. Repeat this process beyond the starting point till the vertex generated last coincides with one of the vertices already generated.

Step 9. The set of coordinates $(x_{jj}\,,\,y_{jj})$ that form a closed chain is the required set of vertices. These are joined in order to determine the polygon.

End.

5.4 Computational complexity

At each point of a digital curve f_j is computed once only except at the points immediately following the vertices where f_j is computed twice, once for the generated line

segment and the second time for the line to be generated. So if the approximation results in m line segments from a closed digital curve with n points ($m \leq n$) then the number of times f_j is computed to generate the approximation is $m + n$. Since $m \leq n$, so $m + n \leq 2n$. As the computation is carried beyond the starting point, a part of the digital curve comes under arithmetic operations twice. But this part is much smaller than the input data size and the additional number of times for which f_j is computed on this part of the digital curve has only additive effect to the sum $m + n$. So the computational complexity of the algorithm is $O(n)$.

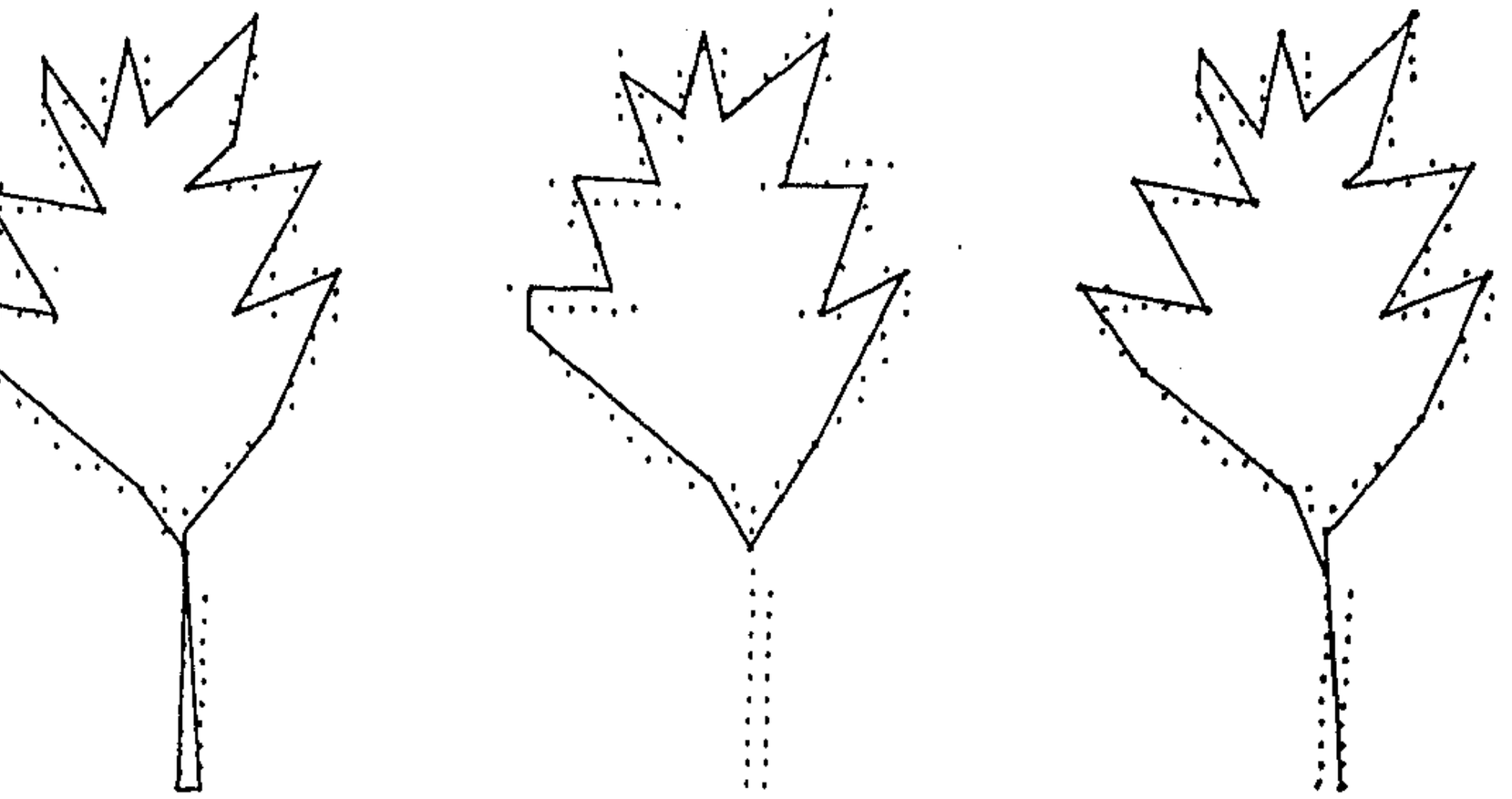
Though the computational complexity is $O(n)$, but the program execution time will be higher if the approximation results in long line segments than that if it results in short line segments. Because in the former case evaluation of s_j involves computation of a large number of terms of the form

$$(x'_k y'_j - y'_k x'_j)^2. \tag{5.9}$$

We illustrate this with the help of an example. Let a closed digital curve consisting of 100 integer coordinate points be approximated by 10 line segments each of which approximates 11 points. Then to generate each line segment evaluation of s_j involves computation of as many as $(10 \times 11)/2 = 55$ terms of the form (5.9). But if another closed digital curve consisting of 100 coordinate points is approximated by 5 line segments, each line segment approximating 21 points then to generate each line segment computation of s_j involves evaluation of $(20 \times 21)/2 = 210$ terms of the form (5.9). So in the former case the total number of terms of the form (5.9) to be evaluated is $55 \times 10 = 550$ and in the later case this figure is $210 \times 5 = 1050$ which is almost twice of the former.

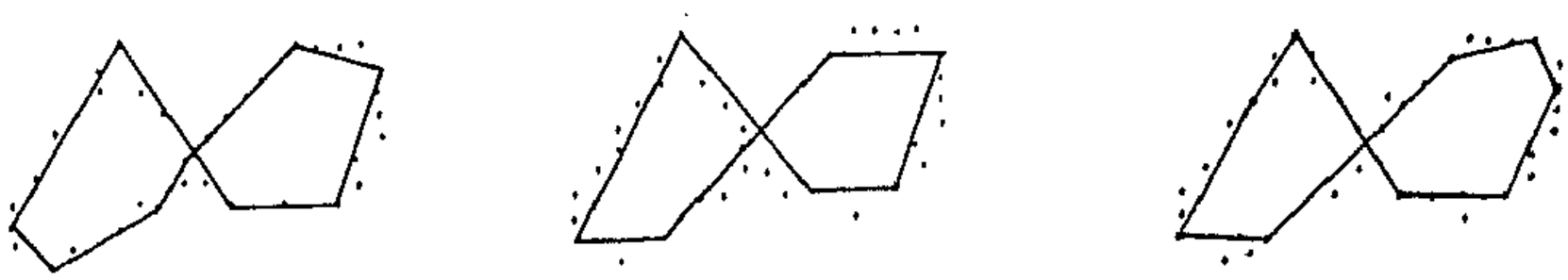
5.5 Experimental results

We have applied the algorithm 5.3 on four digital curves as in the previous chapters. The data are processed in the clockwise direction starting from an arbitrary point. The polygonal approximations are shown in Figures 5.1 through 5.4. Since this procedure is sequential and one-pass hence we compare it with the Williams' algorithm [59] and the Wall-Danielsson algorithm [58]. In Williams' algorithm it is necessary



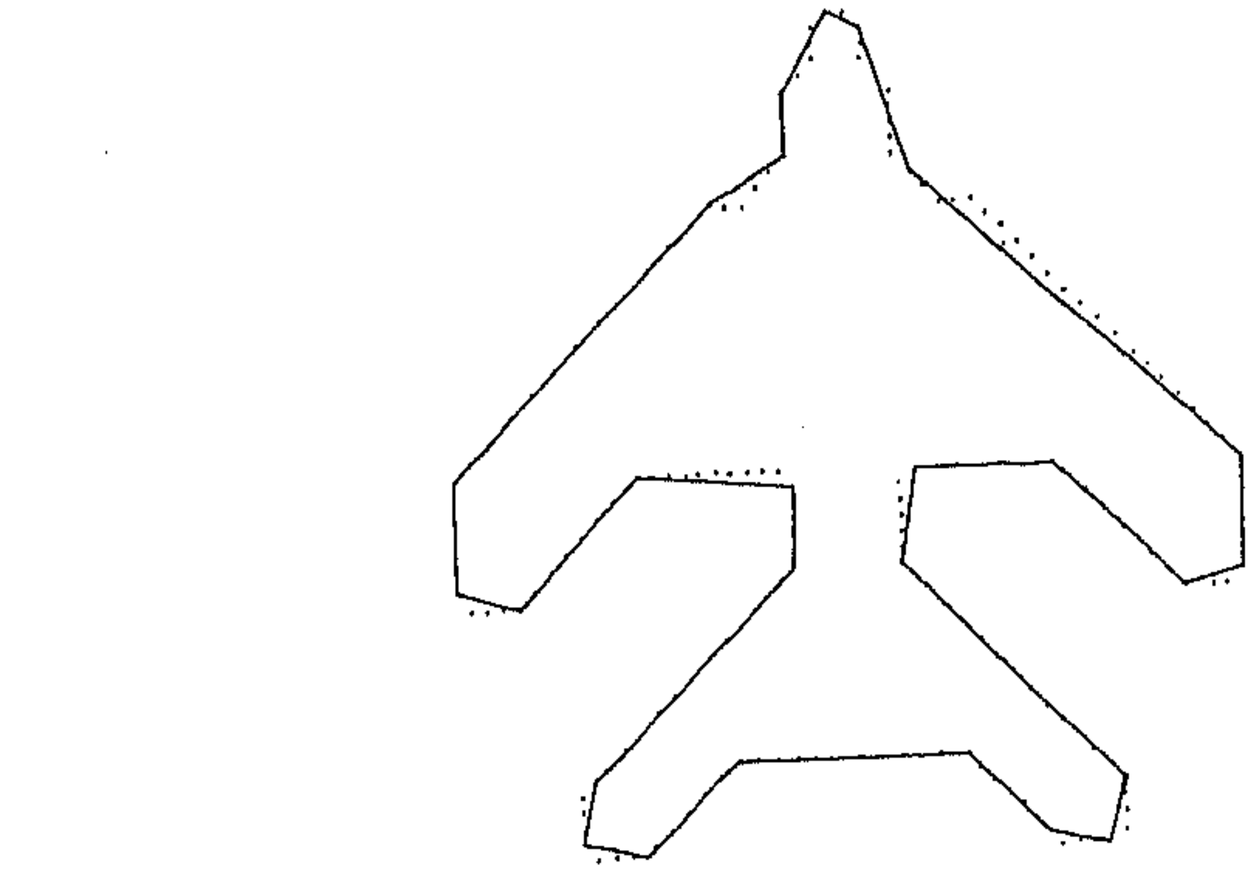
(a) (b) (c)

Figure 5.1: Polygonal approximations of the leaf-shaped curve. (a) Algorithm 5.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

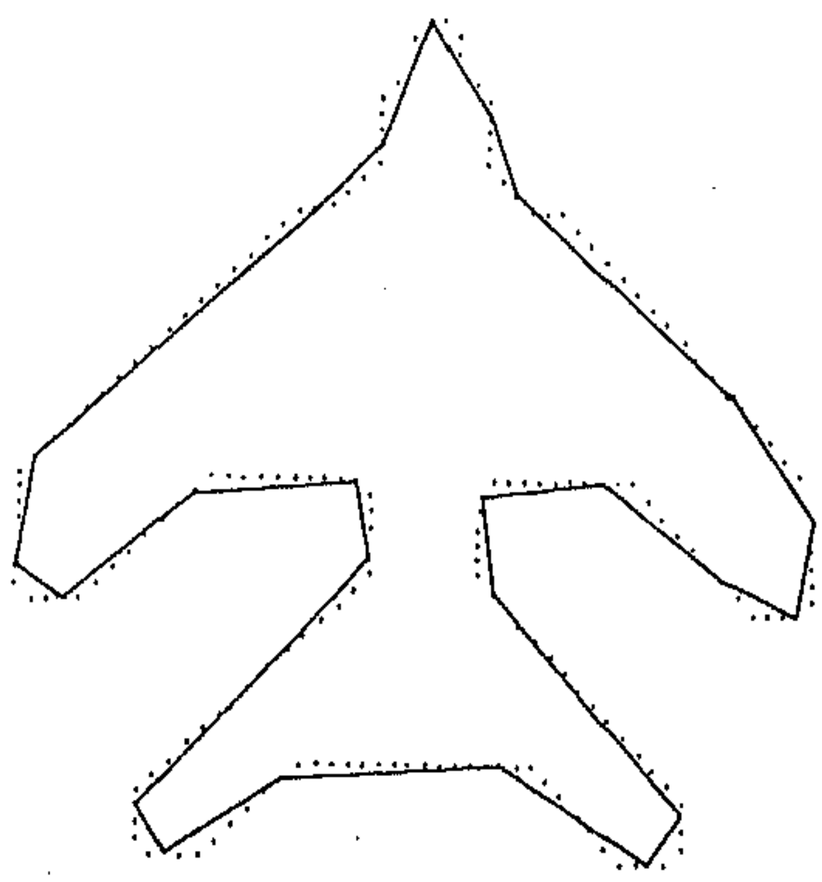


(a) (b) (c)

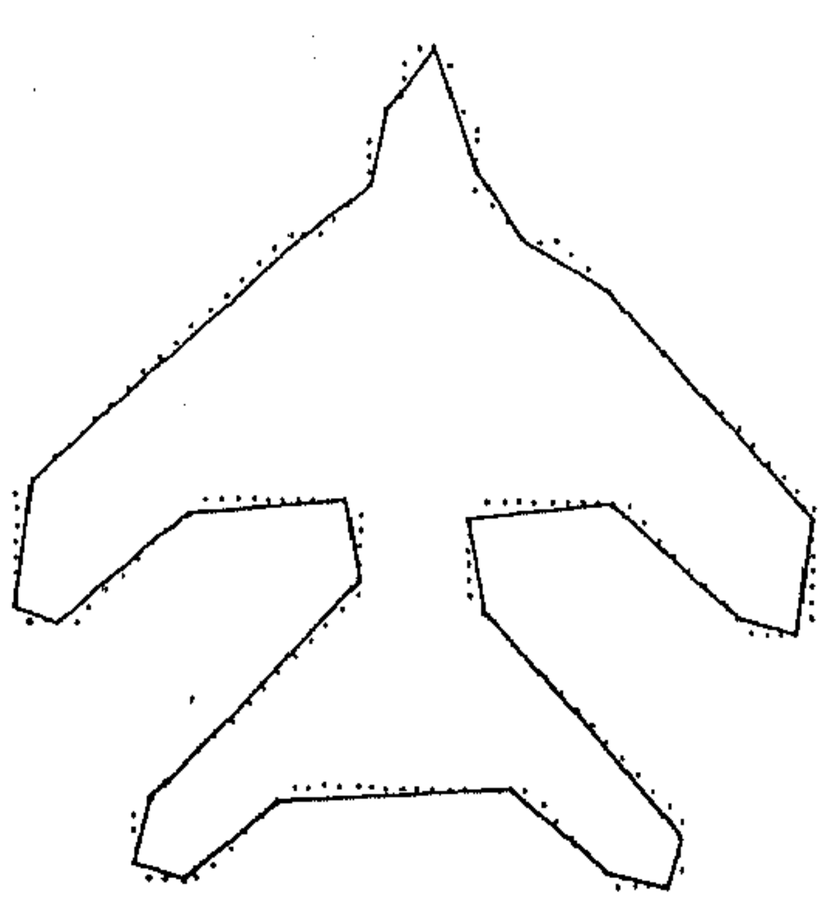
Figure 5.2: Polygonal approximations of the figure-8 curve. (a) Algorithm 5.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)

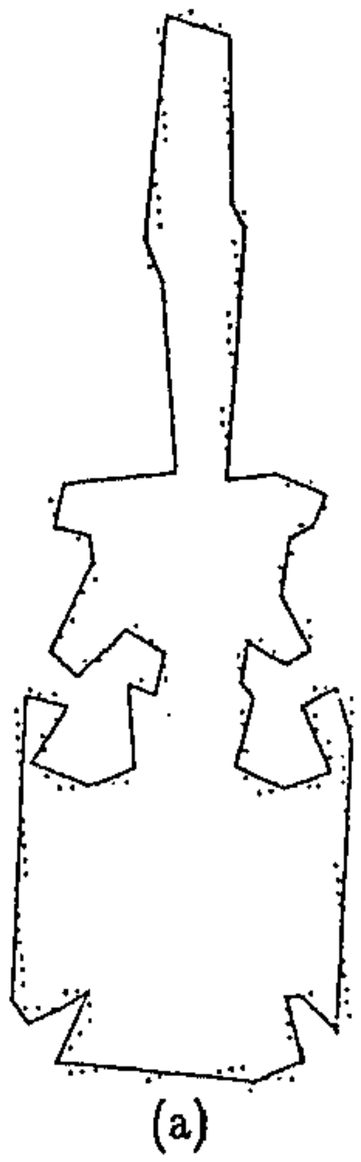


(b)

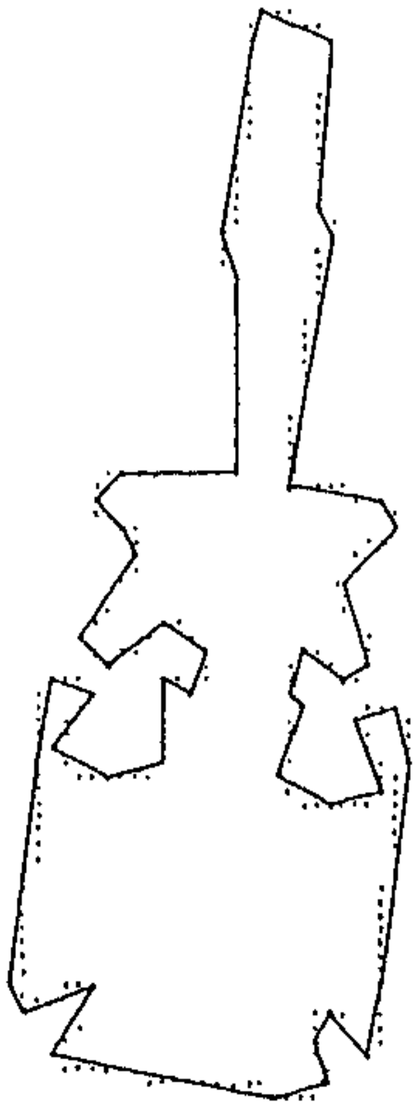


(c)

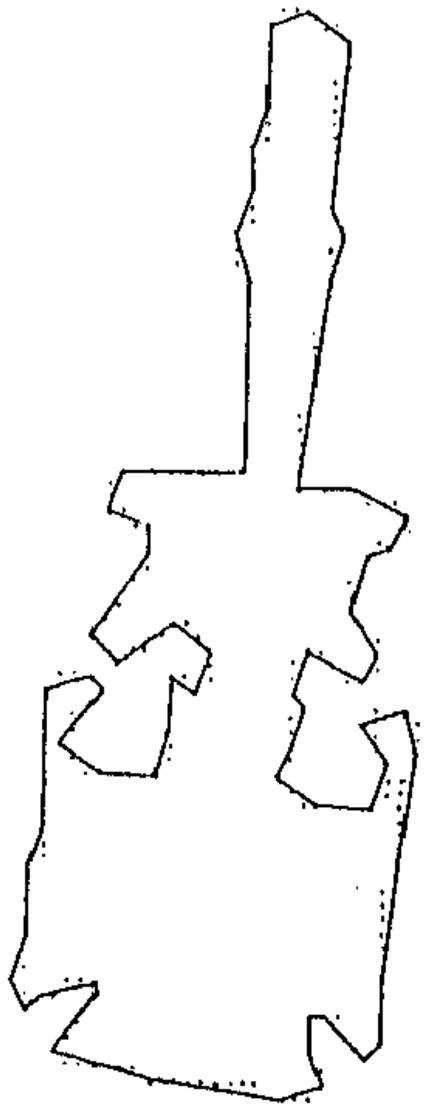
Figure 5.3: Polygonal approximations of the aircraft. (a) Algorithm 5.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)



(b)



(c)

Figure 5.4: Polygonal approximations of the screwdriver. (a) Algorithm 5.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

to specify the maximum allowable error. As already stated we do not specify the maximum allowable error in algorithm 5.3. We keep this error unspecified and allow the procedure to determine it on the basis of the local properties of the curve. The maximum error returned by the algorithm 5.3 is then used to run Williams' algorithm and the Wall-Danielsson algorithm. The polygonal approximations as obtained by the Williams' algorithm and the Wall-Danielsson algorithm are shown in Figures 5.1 through 5.4. It is clear from these figures that the algorithm 5.3 produces better results than the Williams' and the Wall-Danielsson algorithm. The worst approximation produced by the Williams' algorithm is obtained with the leaf-shaped curve, where the lower part of the digital curve, which has a sharp turning, is completely wiped out by the approximation. Moreover, none of the other turnings are approximated according to their nature. The vertices are always placed beyond the turning point. This is true for other digital curves also. Though, algorithm 5.3 is also sequential and one-pass but neither does it round off sharp turnings nor does it shift the vertices from their actual position. The approximations obtained by the Wall-Danielsson algorithm are no better than those by our procedure. It produces redundant / false vertices. It also dislocates the vertices. The peak-test introduces by Wall and Danielsson succeeds to retain the very sharp turnings but it fails to locate the vertices at their actual position when the turnings are not so sharp. An overview of the results of the approximations obtained by the three procedures are displayed in Table 5.1.

Table 5.1 A comparison among algorithm 5.3, Williams' & Wall-Danielsson algorithm				
Digital curve	Leaf	Figure-8	Aircraft	Screwdriver
Number of points (n)	120	45	200	267
Results of algorithm 5.3				
Number of vertices (n_v)	22	9	26	50
Compression ratio (n/n_v)	5.45	5.00	7.69	5.34
Percentage of data reduction	81.67	80.00	87.00	81.27
Integral square error	16.95	6.01	22.80	42.85
Maximum error	1.05	1.00	1.20	0.93
Results of Williams' algorithm				
Maximum allowable error	1.05	1.00	1.20	0.93
Number of vertices (n_v)	18	7	24	50
Compression ratio (n/n_v)	6.67	6.43	8.33	5.34
Percentage of data reduction	85.00	84.44	88.00	81.27
Maximum error	1.05	1.00	1.20	0.91
Integral square error	34.27	14.05	63.33	47.85
Results of Wall-Danielsson algorithm				
Maximum allowable error	1.05	1.00	1.20	0.93
Threshold	0.996	0.91	1.05	0.67
Number of vertices(n_v)	21	9	27	66
Compression ratio(n/n_v)	5.71	5.00	7.41	4.05
Percentage of data reduction	82.50	80.00	86.5	75.24
Maximum error	0.995	1.00	1.05	0.77
Integral square error	21.08	7.08	42.69	25.18

Chapter 6

Another data-driven method

6.1 L_1 norm as a measure of closeness

While approximating a curve by a polygon it is necessary to have a measure of closeness. The error norms are used as a measure of closeness. The most commonly used norms are the maximum error (L_∞ norm) and the integral square error (L_2 norm). The existing algorithms for polygonal approximation of digital curve use either the maximum error or the integral square error. In the polygonal approximation schemes where the approximation errors are controlled indirectly, the maximum error is controlled by the criterion function. In this chapter we wish to show that though the most commonly used norms are integral square error and the maximum error but it is possible to use the sum of absolute errors (L_1 norm) as a measure of closeness. The procedure [43] that we present here is conceptually and technically similar to that presented in the last chapter.

6.2 Procedure

The equation of a straight line joining the point p_i to the point p_j is

$$(y_j - y_i)x - (x_j - x_i)y - x_i y_j + x_j y_i = 0. \tag{6.1}$$

Translating the origin to the point $(x_i\ ,\ y_i)$ and denoting the new coordinate system by prime so that

$$x'_j = x_j - x_i \quad and \quad y'_j = y_j - y_i \tag{6.2}$$

the equation (6.1) reduces to

$$y'_j x' - x'_j y' = 0. \tag{6.3}$$

The error e_k between the point p_k , $k = i+1, i+2, ..., j-1$ and the line (6.3) is the perpendicular distance of the point p_k from the line (6.3) that is,

$$e_k = \frac{y'_j x'_k - x'_j y'_k}{\sqrt{x'^2_j + y'^2_j}}. \tag{6.4}$$

So while approximating the points p_k , $k = i, i+1, i+2, ..., j$; the sum of absolute errors along the line segment joining the point p_i to the point p_j is

$$abs\ s_j = \sum_{k=i+1}^{j-1} |e_k|. \tag{6.5}$$

And the length of the line segment joining p_i to p_j is

$$l_j = \sqrt{x'^2_j + y'^2_j}. \tag{6.6}$$

Our objective is to make l_j as large as possible so that $abs\ s_j$ is as small as possible. Note that neither we can increase l_j indefinitely nor we can diminish the sum $abs\ s_j$ arbitrarily. Because, in the former case, the approximation error will be too large resulting in an approximation that may fail to locate many significant vertices. And in the later case, the approximation may result in many unwanted vertices. So it is necessary to make a compromise between the length of the line segment and the approximation error. We solve this problem by combining l_j with $abs\ s_j$. Since our object is to make l_j as large as possible so that $abs\ s_j$ is as small as possible, so we take the mathematical combination

$$f_j = l_j - abs\ s_j\ . \tag{6.7}$$

The procedure looks for the local maxima of f_j by continuously merging points one after another with the point p_i. The value of j for which f_j attains a local maximum gives the location of a vertex. The procedure can be started from an arbitrary point and is carried beyond the starting point till the vertex generated last coincides with one of the vertices already generated.

6.3 Algorithm

The input are the data points $(x_i\,,\,y_i)$, $i = 1, 2, ..., n$. The output are the vertices $(x_{jj}\,,\,y_{jj})$ and jj. All arithmetics are in modulo n.

Begin

Step 1. Initiate $i = 1$.

Step 2. Translate the origin of the coordinate system to the point $(x_i\,,\,y_i)$ by the transformation rule

$$x'_j = x_j - x_i \quad \textit{and} \quad y'_j = y_j - y_i\,.$$

Step 3. Set $j = i + 1$.

Step 4. Compute $f_j = \sqrt{x'^{\,2}_j + y'^{\,2}_j}$.

Step 5. Change j to $j + 1$.

Step 6. Compute $l_j = \sqrt{x'^{\,2}_j + y'^{\,2}_j}$.

Step 7. Compute

$$e_k = \frac{x'_k y'_j - x'_j y'_k}{\sqrt{x'^{\,2}_j + y'^{\,2}_j}}.$$

Step 8. Compute $f_j = l_j - abs\ s_j$.

Step 9. If $f_j \geq f_{j-1}$ *then go to* Step 5

else write $jj = j - 1$ and $(x_{jj}\,,\,y_{jj})$ using

$$x_{jj} = x'_j + x_i \quad \textit{and} \quad y_{jj} = y'_j + y_i\,;$$

set $i = jj$; go to Step 2.

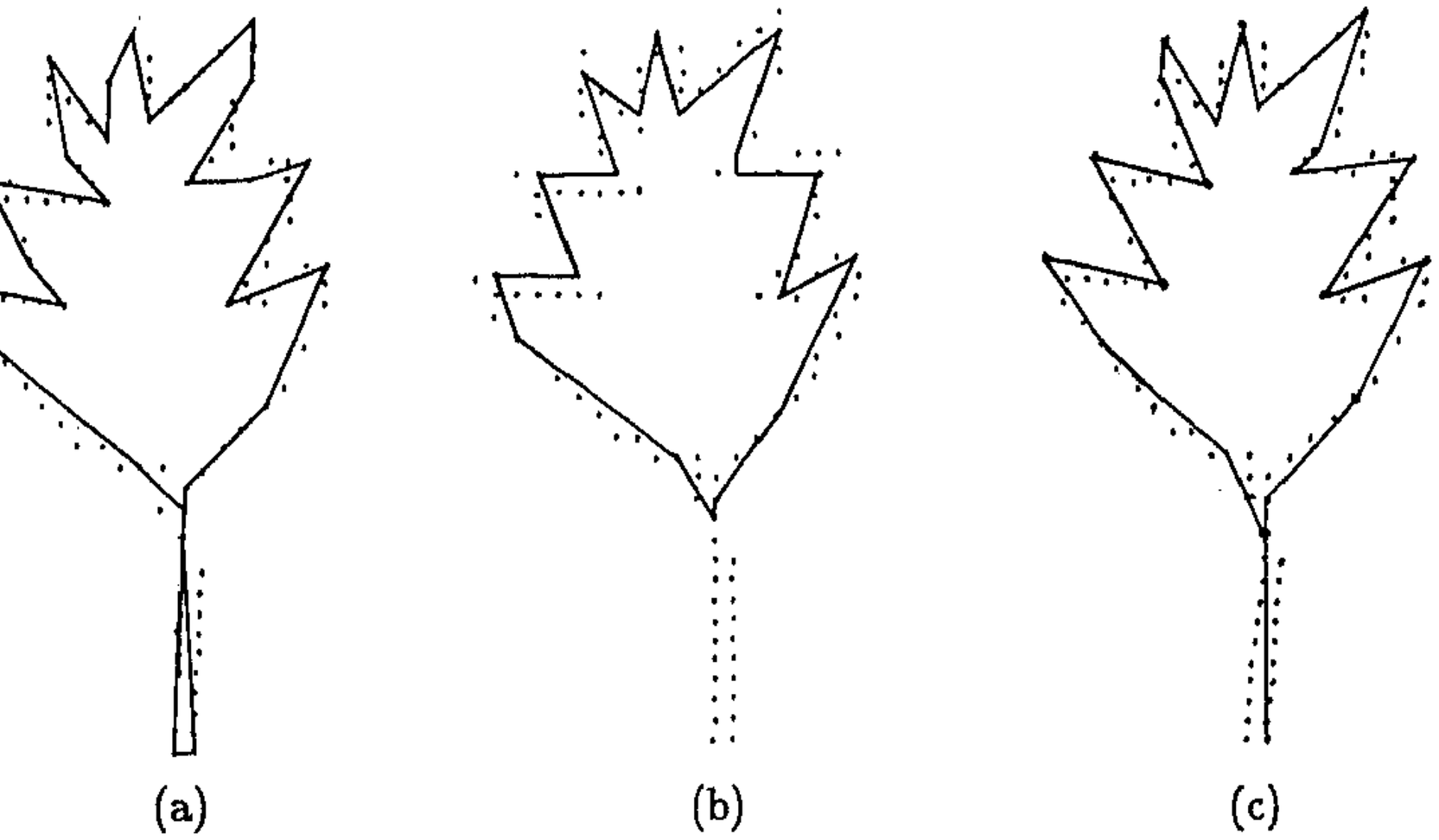
Step 10. Repeat this process beyond the starting point till the the vertex generated last coincides with one of the vertices already generated.

Step 11. The set of coordinates $(x_{jj}\,,\,y_{jj})$ that form a closed chain is the required set of vertices. These are joined in order to determine the polygon.

End.

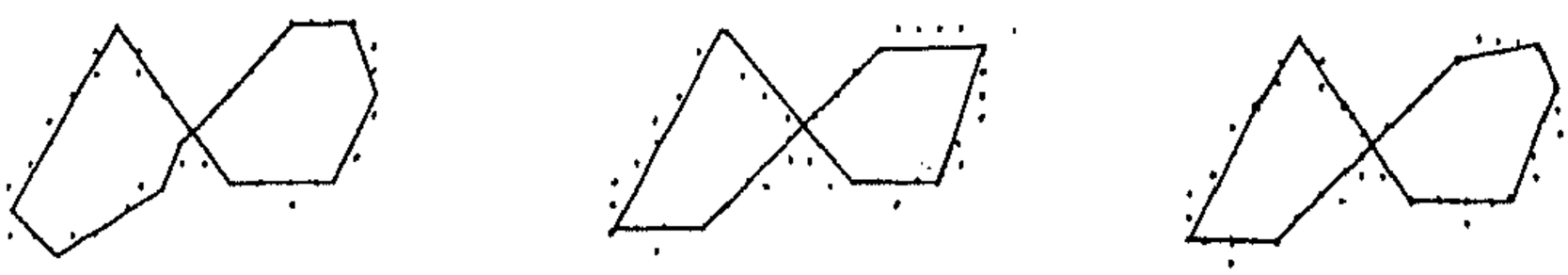
6.4 Computational complexity

The procedure is sequential one-pass and following the same line of arguments as presented in the last chapter it can be derived that the computational complexity



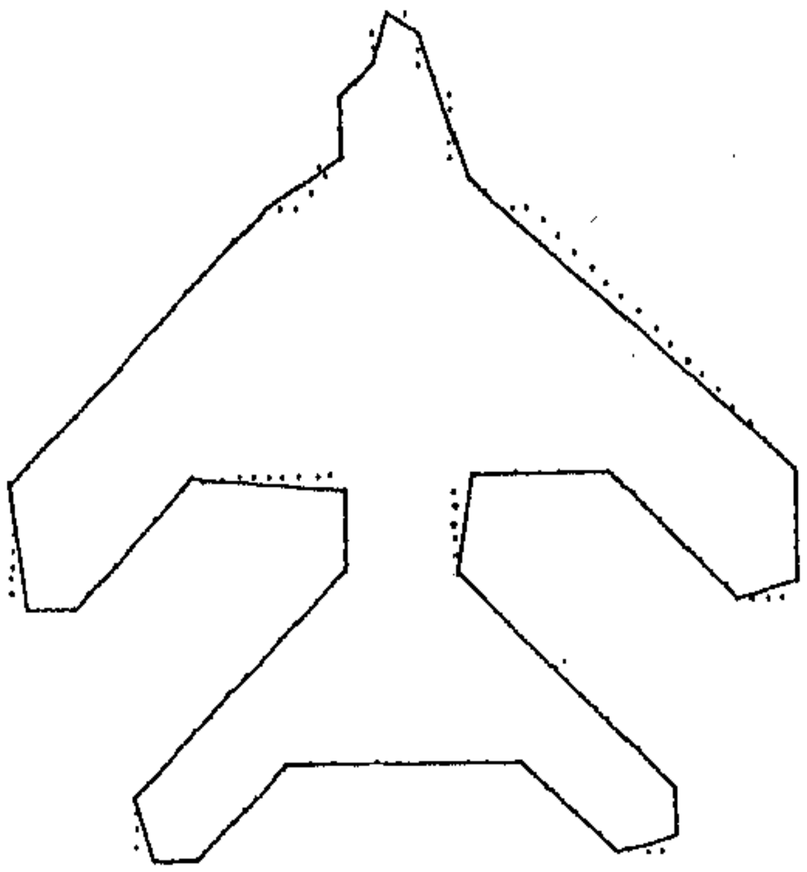
(a) (b) (c)

Figure 6.1: Polygonal approximations of the leaf-shaped curve. (a) Algorithm 6.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

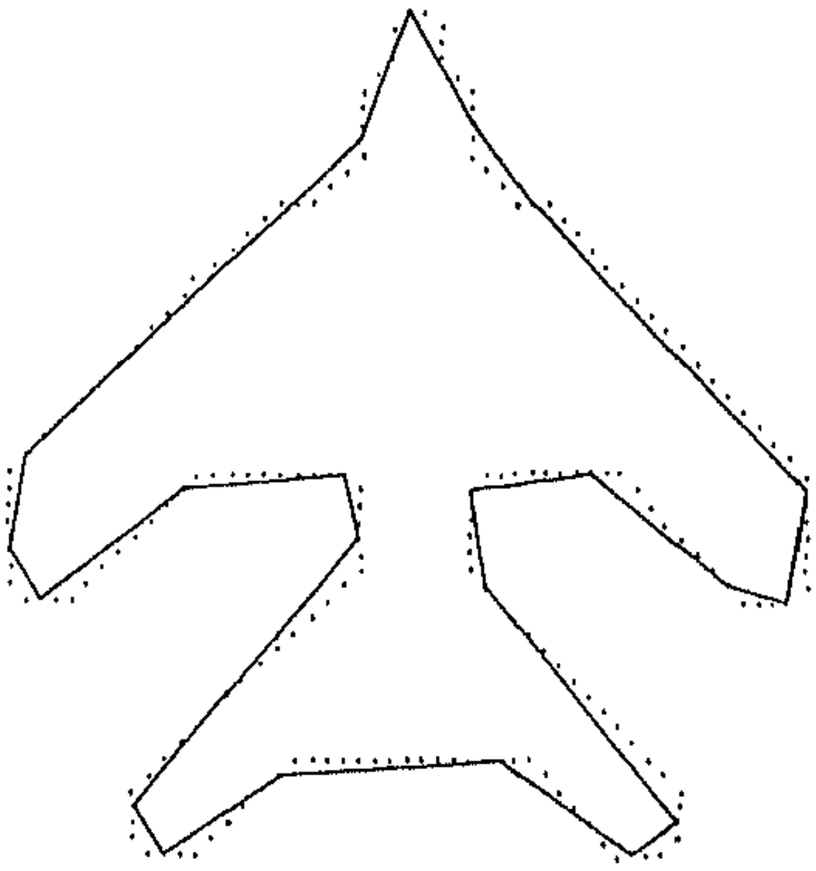


(a) (b) (c)

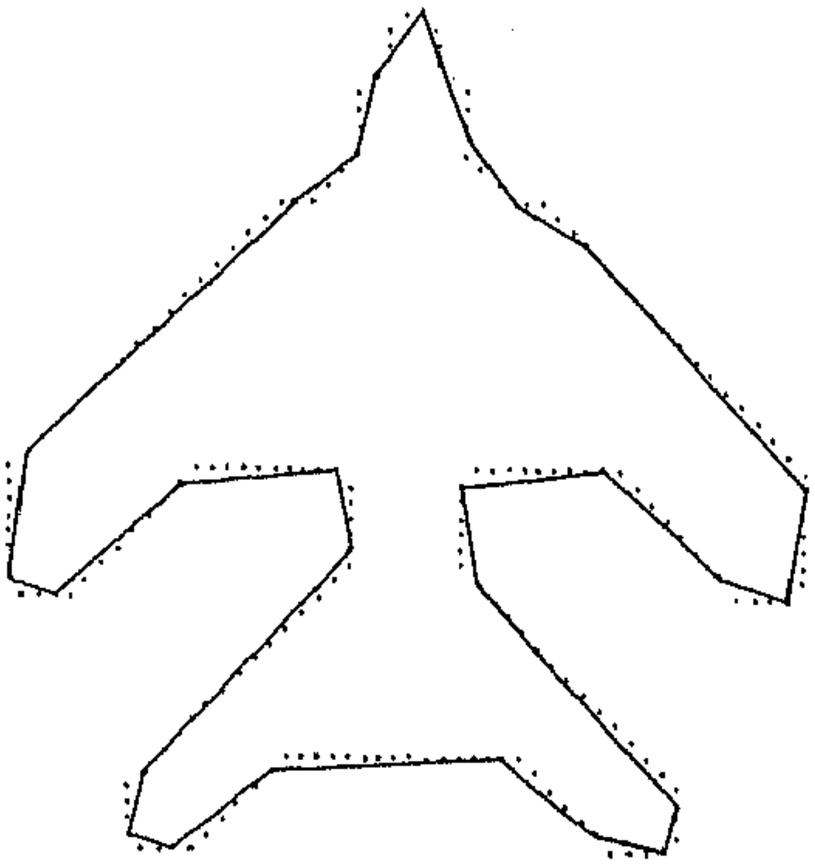
Figure 6.2: Polygonal approximations of the figure-8 curve. (a) Algorithm 6.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)

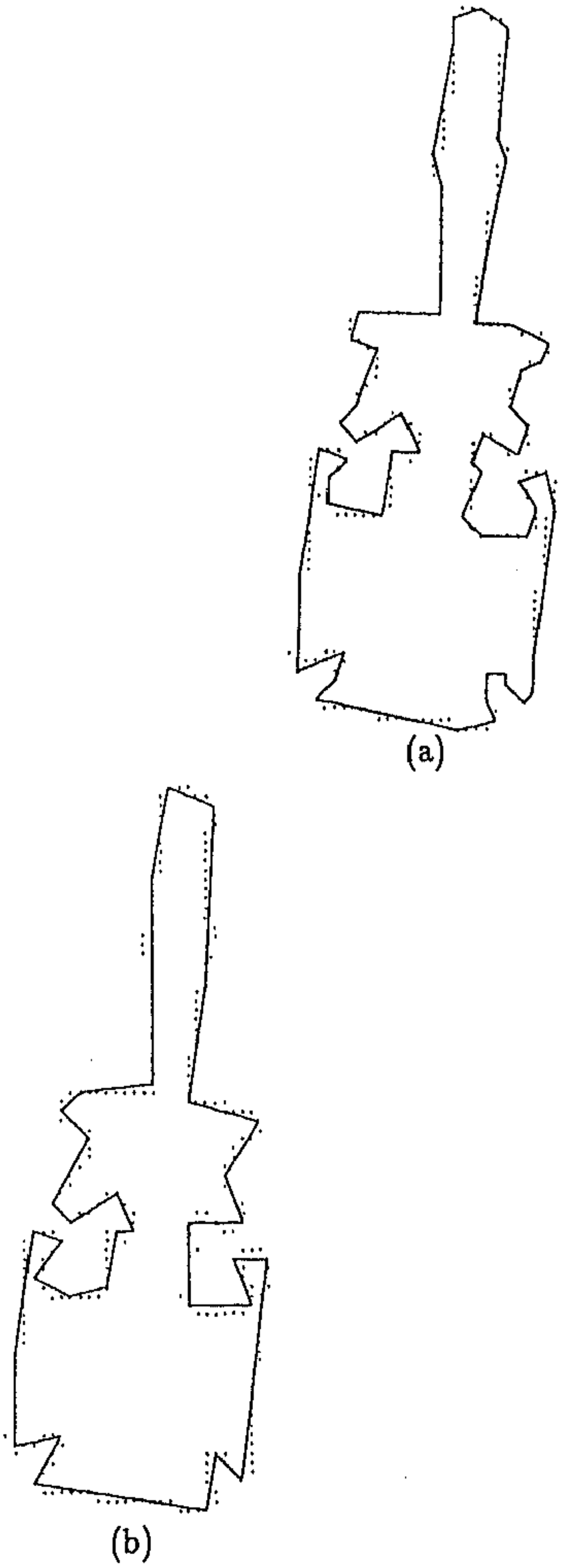


(b)



(c)

Figure 6.3: Polygonal approximations of the aircraft. (a) Algorithm 6.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.



(a)

(b)

(c)

Figure 6.4: Polygonal approximations of the screwdriver. (a) Algorithm 6.3, (b) Williams' algorithm, (c) Wall-Danielsson algorithm.

of the algorithm is $O(n)$ and the program execution time will be higher if the approximation results in long line segments than that if it results in short line segments.

6.5 Experimental results

We have applied algorithm 6.3 on four digital curves as in the previous chapters. The data are processed in the clockwise direction. The procedure is compared with the Williams' [59] and the Wall-Danielsson [58] algorithm. The polygonal approximations as obtained by algorithm 6.3, the Williams' algorithm and the Wall-Danielsson algorithm are shown are shown in Figures 6.1 through 6.4. As seen from these figures the approximations generated by our procedure do not round off sharp turnings nor do they miss corners. The approximations produced by the Williams' algorithm round off sharp turnings and miss corners. The approximations produced by the Wall-Danielsson algorithm are no better than our procedure. The results of the approximations as obtained by the three procedures are shown in Table 6.1.

Table 6.1 A comparison among algorithm 6.3, Williams' & Wall-Danielsson algorithm				
Digital curve	Leaf	Figure-8	Aircraft	Screwdriver
Number of *points*(n)	120	45	200	267
Results of algorithm 6.3				
Number of vertices(n_v)	26	10	28	56
Compression ratio(n/n_v)	4.62	4.5	7.14	4.77
Percentage of data reduction	78.33	77.80	86.00	79.03
Sum of absolute error	29.43	8.62	34.27	68.70
Maximum error	1.05	1.0	1.33	1.00
Results of Williams' algorithm				
Maximum allowable error	1.05	1.00	1.33	1.00
Number of vertices(n_v)	18	7	22	36
Compression ratio(n/n_v)	6.67	6.43	9.09	7.42
Percentage of data reduction	85.00	84.44	89.00	86.50
Maximum error	1.05	1.00	1.31	1.00
Sum of absolute error	45.62	17.96	105.54	101.25
Results of Wall-Danielsson algorithm				
Maximum allowable error	1.05	1.00	1.33	1.00
Threshold	0.996	0.91	1.05	0.78
Number of vertices(n_v)	21	9	27	61
Compression ratio(n/n_v)	5.71	5.00	7.41	4.38
Percentage of data reduction	82.50	80.00	86.50	77.13
Maximum error	0.995	1.00	1.05	1.00
Sum of absolute error	38.88	11.48	75.94	57.84

Chapter 7

Polygonal approximation as angle detection

7.1 Curvature based polygonal approximation

The algorithms developed in the last chapters treat polygonal approximation as a side detection problem. The sides of the polygon are determined subject to certain constraints on the goodness of fit. Another approach to polygonal approximation is based on curvature estimation.

In the real Euclidean plane, curvature (κ) is defined as the rate of change of tangential angle (ψ) with respect to the arc length (s)

$$\kappa = \frac{d\psi}{ds}. \tag{7.1}$$

In Cartesian coordinate system if the equation of a curve is expressed in the form $y = f(x)$, then the curvature at any point of the curve is defined by

$$\kappa = \frac{\frac{d^2y}{dx^2}}{\left\{1 + \left(\frac{dy}{dx}\right)^2\right\}^{\frac{3}{2}}}. \tag{7.2}$$

For digital curves however, it is not immediately clear how to define a discrete analog of curvature. If the discrete curvature is defined by simply replacing the derivatives in (7.2) by finite differences, there is a problem that small changes in

slope are impossible, since the successive slope angles on a digital curve can differ only by a multiple of $\pi/4$. This difficulty is avoided by introducing a smoothed version of discrete curvature e.g. k−cosine, k−curvature, k is called the region of support and it is determined by introducing an input parameter m. The input parameter m is selected on basis of the level of detail of the curve. The finer is the level of detail, the smaller should be the value of m. Difficulty arises when a curve consists of features of multiple size. And this difficulty can only be avoided by using different m for regions with different levels of detail. Moreover, there is seldom any basis of choosing a particular value of parameter for a particular feature size. Teh and Chin [56] used chord length and perpendicular distance to determine the region of support without using any input parameter. This technique is justified, because the region of support for each point is determined only on the basis of the local properties of the curve.

In this chapter we present an algorithm [44] for polygonal approximation of digital curves which is based on discrete curvature measure. We propose that one can use k−cosine itself to determine the region of support without using any input parameter. A new measure of discrete curvature based on k−cosine, called smoothed k−cosine, is introduced. The local maxima and minima of smoothed k−cosine are located. We call these points as *significant* points. The adjacent significant points are joined to determine the polygon.

In the following section we present a scheme to determine the region of support of each point of a digital curve. The procedure is parallel in the sense that the results obtained at each point do not depend on those obtained at the other points. The procedure need no input parameter. The region of support of each point is determined on the basis of the local properties of the curve.

7.2 Procedure:Determination of region of support

Begin

Step 1. Define the k−vectors at p_i as

$$\mathrm{a}_{ik} = (x_{i-k} - x_i\ ,\ y_{i-k} - y_i) \tag{7.3}$$

$$\mathbf{b}_{ik} = (x_{i+k} - x_i\,,\; y_{i+k} - y_i) \tag{7.4}$$

and the k–cosine at p_i as

$$\cos_{ik} = \frac{\mathbf{a}_{ik} \,.\, \mathbf{b}_{ik}}{|\,\mathbf{a}_{ik}\,|\;|\,\mathbf{b}_{ik}\,|} \tag{7.5}$$

where $\cos_{ik}$ is the cosine of the angle between the k–vectors $\mathbf{a}_{ik}$ and $\mathbf{b}_{ik}$, so that $-1 \leq \cos_{ik} \leq +1$.

Step 2. Start with $k = 1$. Compute $\cos_{ik}$ giving increment to k.

If $|\cos_{i,k+1}| > |\cos_{ik}|$ then k determines the region of support of p_i

else if $|\cos_{ik}| = |cos_{i,k+1}|$ then the greatest k for which this relation holds determines the region of support of p_i

else if $\cos_{ik}$ and $\cos_{i,k+1}$ be of opposite sign then the least value of k for which it happens gives the region of support of p_i.

The region of support of p_i is the set of points given by

$$D(p_i) = \{\, p_{i-k}, \ldots, p_{i-1}, p_i, p_{i+1}, \ldots, p_{i+k} \,\}.$$

End.

7.3 Measure of significance

The last procedure determines the region of support (k_i) of the point p_i. To detect the significant points we need a measure of significance. Rosenfeld and Johnston [48] used $\cos_{ih_i}$ as the measure of significance and h_i as the region of support of p_i. Rosenfeld and Weszka [50] used smoothed k cosine as a measure of significance. We propose to introduce a new measure of significance. We denote it by $\cos_i$ and define it by

$$\cos_i = \frac{1}{k_i} \sum_{j=1}^{k_i} \cos_{ij}\ . \tag{7.6}$$

This measure of significance is a kind of smoothed cosine but it is different from that given by Rosenfeld and Weszka [50]. In the following section we present a procedure for detection of significant points using the region of support k determined by the procedure given in section 7.2 and the measure of significance $\cos_i$ introduced in this section.

7.4 Procedure:Detection of significant points and polygonal approximation

Comments. As the procedure runs remove those points from consideration where $\cos_i$'s are too small ($\cos_i \leq -0.800$), because in the neighborhood of these points the curves are relatively straight and our ultimate goal is to make a polygonal approximation of the curves.

Begin

1st pass. Retain only those points p_i for which either

$$(a)\ \cos_i \geq \cos_j \tag{7.7}$$

for all j satisfying

$$\begin{aligned} |i-j| &\leq \frac{k_i}{2}, \quad k_i > 1 \\ &= k_i, \quad k_i = 1. \end{aligned} \tag{7.8}$$

or,

$$\cos_i \leq \cos_j \tag{7.9}$$

for all j satisfying

$$\begin{aligned} |i-j| &\leq \frac{k_i}{2}, \quad k_i > 1 \\ &= k_i, \quad k_i = 1. \end{aligned} \tag{7.10}$$

In (7.7) and (7.9) strict inequality should hold for at least one j satisfying (7.8) and (7.10) respectively. The points detected by (7.7) are the local maxima and those detected by (7.9) are the local minima of smoothed $k-$cosine.

2nd pass. If a minima point falls within the region of support of a maxima point then the minima point is discarded and the maxima point is retained.

If two successive points p_i and p_{i+1} appear as maxima points then

if both p_i and p_{i+1} have the same cosine and the same region of support then retain p_i and discard p_{i+1}

else if the cosines be the same but the region of support be different then retain the point with higher region of support and discard the other.

The points obtained from the 1st and the 2nd pass constitute the set of significant points and the adjacent significant points are joined to make a polygonal approximation.

End.

Remark 1. When two successive points p_i and p_{i+1} appear as maxima points and both have the same cosine then both the points are equally important for being selected as significant points and so in this situation there is a tie. We have proposed to break the tie by choosing p_i only as the significant point. On the other hand, when p_i and p_{i+1} have the same cosine but different region of support then there is no tie and the choice is deterministic.

Remark 2. The 1st pass of the procedure can be carried out in parallel whereas, the 2nd pass is sequential. We note that the 2nd pass is carried out only on a small number of points.

7.5 Approximation errors

The shape of a digital curve is determined by its significant points. So it is very much necessary to locate them accurately so that sufficient information about the curve is contained in the location of the significant points. In section 7.2 and section 7.4 we have described a procedure to detect significant points of digital curves.

We propose to measure the accuracy of the location of the significant points by the pointwise error between the digital curve and the approximating polygon. We measure the error between the digital curve and the approximating polygon by the perpendicular distance of the point p_i's from their approximating line segment. We denote this error by e_i. Two error norms between the digital curve and the approximating polygon are

1) Integral square error $E_2 = \sum_{i=1}^{n} {e_i}^2$,

2) Maximum error $E_\infty = \max_i \mid e_i \mid$.

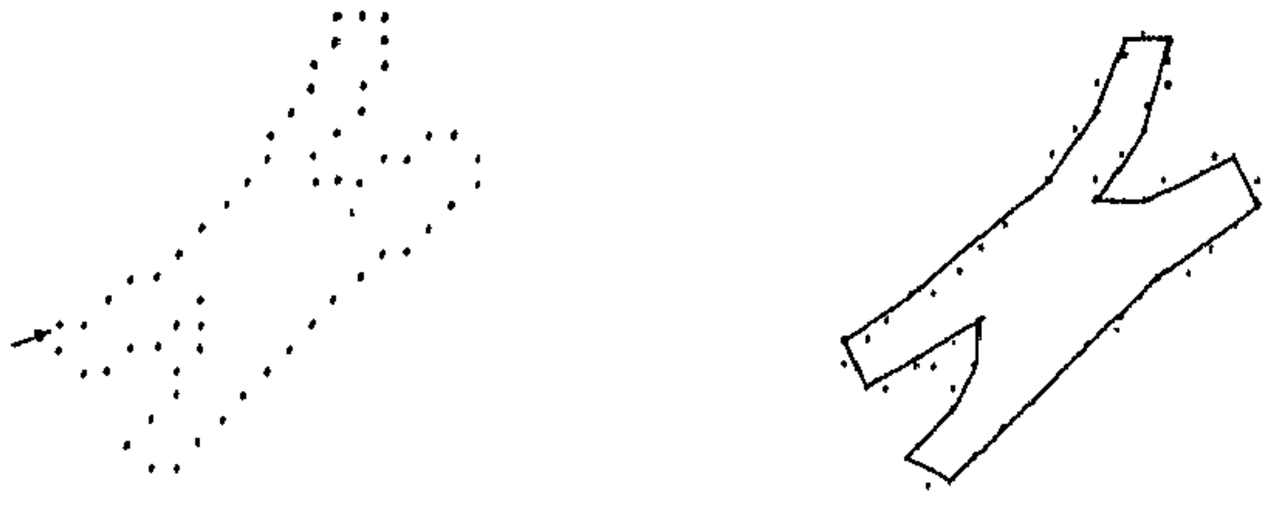


Figure 7.1: A chromosome shaped curve and its polygonal approximation.

Figure 7.2: The figure-8 curve and its polygonal approximation.

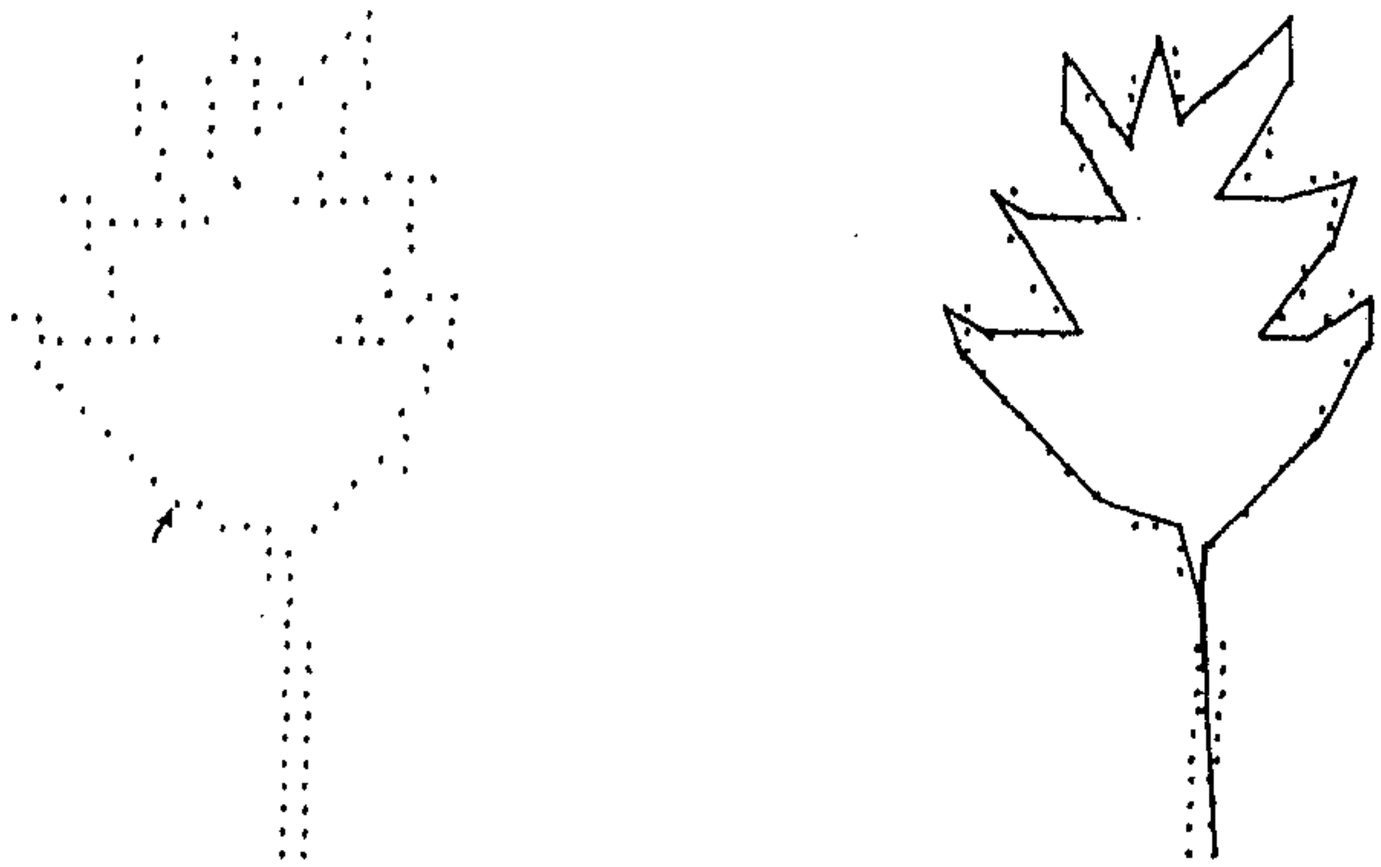


Figure 7.3: The leaf-shaped curve and its polygonal approximation.

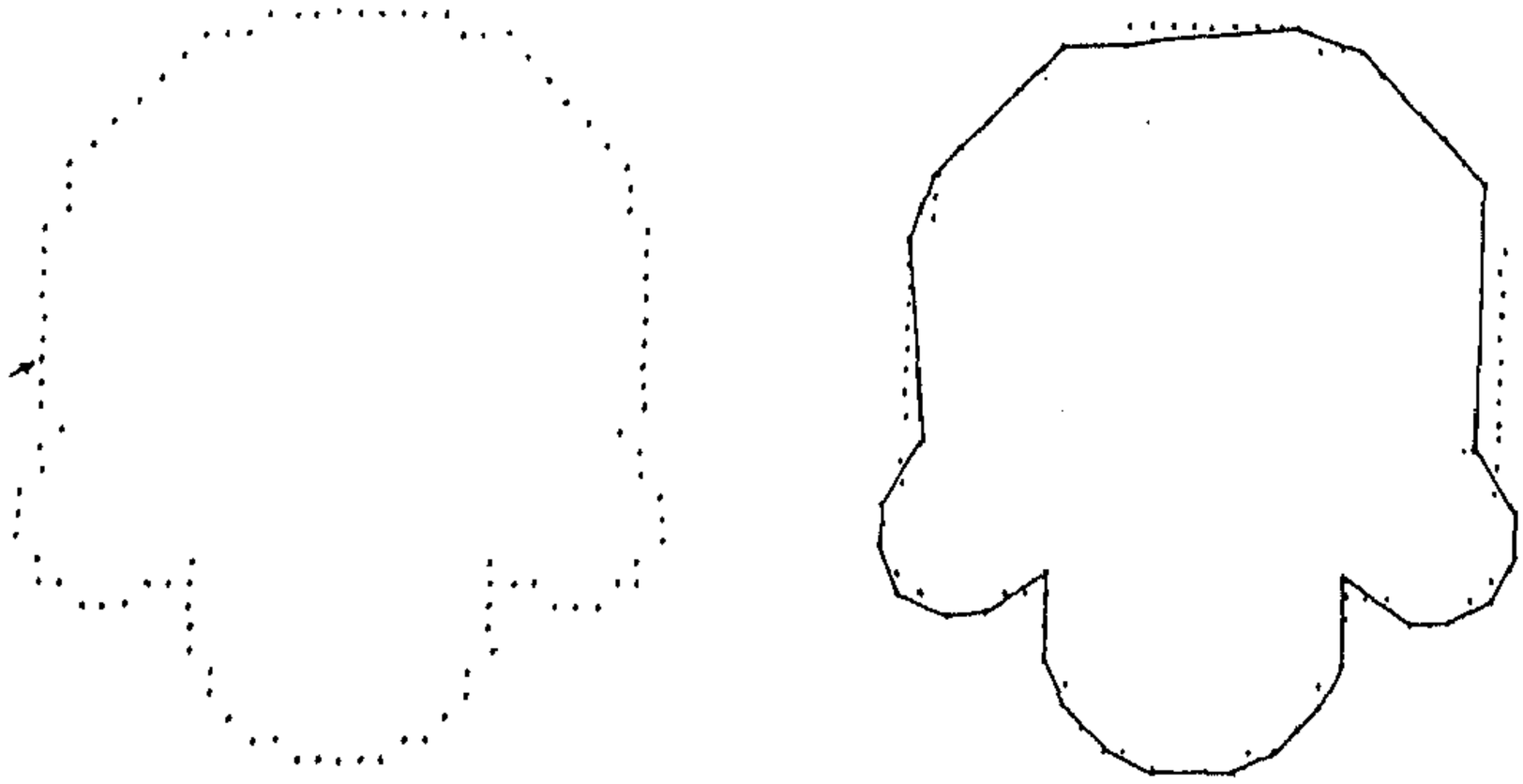


Figure 7.4: A curve with four semi-circles and its polygonal approximation.

7.6 Experimental results

The procedure presented in this chapter is applied on four digital curves namely, a chromosome shaped curve (Figure 7.1), a figure-8 curve (Figure 7.2), a leaf-shaped curve (Figure 7.3), and a curve with four semi-circles (Figure 7.4). The last figure, taken from [56], is an example of a curve that consists of features of multiple size. The chain code of the curves are given in Table 7.1. The curves have been coded in the clockwise direction starting from the point marked with an $\nearrow$ on each curve, using the Freeman chain code defined in chapter 1. The procedure processes data in the clockwise direction.

In an attempt to focus on the efficiency of our procedure as a significant point detector and a polygonal approximation technique we have computed (a) the data compression ratio given by n/n_s, n_s being the number of significant points, (b) the integral square error and (c) the maximum error. The results are displayed in Table 7.2, together with the results obtained by the Teh and Chin algorithm [56] (reproduced from [56]).

7.7 Discussion

We have made no attempt of comparing our procedure with those in [48], [50], [17], [2] and [53]. A very good comparison of these algorithms with that in [56] can be found in [56]. We only focus on the following features of our procedure.

(a) Our procedure, like that of Sankar-Sharma [53] and Teh and Chin [56], does not require any input parameter.

(b) Sankar-Sharma's procedure does not require any input parameter but it does not determine region of support, whereas, our procedure, like that of Teh and Chin, does determine region of support.

(c) Our procedure detects more significant points than the Teh-Chin algorithm.

(d) All processing are done locally, so it is be suitable for parallel processing.

Table 7.1 Chain code of digital curves							
Chromosome shaped curve							
01101	11112	11212	00665	65560	01010	76555	45555
55555	43112	12255	45432				
Figure-8 curve							
11217	67767	70071	01212	22344	45555	56545	54534
22112							
Leaf-shaped curve							
33333	32307	00003	32323	07000	03323	22267	77222
12766	61111	16665	66550	00100	56656	55001	10665
65655	55566	67666	66666	64222	22222	22232	24434
Curve in figure 7.4							
22222	21221	11111	00100	00000	07007	77777	66766
66666	65767	66564	54434	36666	56554	54444	34332
32222	54544	34232	21213	22			

Table 7.2 A comparison between algorithm 7.4 & Teh-Chin algorithm				
Digital curve	Chromosome	Figure-8	Leaf	Fig. 7.4
Number of points(n)	60	45	120	102
Results of algorithm 7.4				
Number of significant points(n_s)	18	14	29	30
Compression ratio(n/n_s)	3.33	3.21	4.14	3.40
Integral square error	5.25	2.51	15.34	17.19
Maximum error	0.686	0.728	0.996	1.00
Results of Teh-Chin algorithm				
Number of dominant points(n_d)	15	13	29	22
Compression ratio(n/n_d)	4.00	3.50	4.10	4.60
Integral square error	7.20	5.93	14.96	20.61
Maximum error	0.74	1.00	0.99	1.00

Chapter 8

Polygonal approximation as angle detection using asymmetric region of support

8.1 Asymmetric region of support

In the last chapter we have shown that it is possible to use k–cosine to determine the region of support without using any input parameter. The region of support as determined by this procedure is symmetric in the sense that it consists of equal number of points on either side of the point of interest. The region of support as determined by Rosenfeld-Johnston [48], Rosenfeld-Weszka [50] and Teh and Chin [56] is also symmetric. But there is no reason why the region of support should be symmetric. We believe that an asymmetric region of support is more reasonable and more natural than a symmetric region of support. A symmetric region of support may be looked upon as a special case of asymmetric region of support.

In this chapter we propose to determine asymmetric region of support without using any input parameter. The region of support is determined on the basis of the local properties of the curve. We call the regions on either side of the point p_i as the arms of the point. The curve is described in the clockwise direction. The arm extending from the point p_i to the forward direction is regarded as the *right* arm of

p_i and the arm extending from the point p_i to the backward direction is regarded as the *left* arm of the point p_i. The size of the arms is the number of points comprising the arms. We propose to denote the size of the right arm by k and that of the left arm by l. The region of support of p_i comprises the points in its arms and the point p_i itself.

This concept of asymmetric region of support is utilized to detect dominant points on digital curves. The dominant points are the points with high curvature. To determine the curvature using the concept of asymmetric region of support we introduce the concept of $k-l$ cosine which is defined as the cosine of the angle between the k–vector and the l–vector at the point p_i. The dominant points are those points at which the $k-l$ cosine is a local maximum with respect to its immediate neighbors. The polygonal approximation is made joining the adjacent dominant points [45].

8.2 Determination of arms

We define two vectors $\mathbf{R}_i$ and $\mathbf{R}_{ij}$ at p_i by

$$\begin{aligned} \mathbf{R}_i &= (x_{i+1}-x_i\,,\,y_{i+1}-y_i) \\ \mathbf{R}_{ij} &= (x_j-x_i\,,\,y_j-y_i) \end{aligned} \tag{8.1}$$

where $j = i+2,\ i+3,\ i+4,\ldots$. If θ_j denotes the angle between $\mathbf{R}_i$ and $\mathbf{R}_{ij}$ (Figure 8.1) then we propose to compute θ_j by the relation

$$\theta_j = \cos^{-1}\left(\frac{\mathbf{R}_i\,.\,\mathbf{R}_{ij}}{|\mathbf{R}_i|\,|\mathbf{R}_{ij}|}\right). \tag{8.2}$$

θ_j is computed giving increment to j until for some j

$$\theta_{j-1} < \theta_j \leq \theta_{j+1}\,. \tag{8.3}$$

When the last relation holds for three consecutive values of j $(j-1, j, j+1)$ then the set of points $\{\,p_{i+1}, p_{i+2}, \ldots, p_{j-1}\,\}$ is said to constitute the *right* arm of the point p_i, and the length of the arm is $k = j-i-1$.

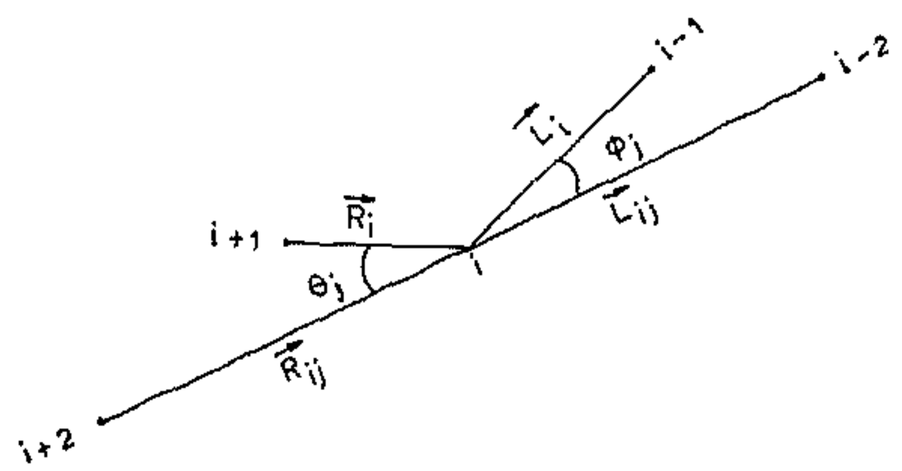


Figure 8.1: Left and right vector and angles θ and ϕ.

Again we define two vectors $\mathbf{L}_i$ and $\mathbf{L}_{ij}$ at p_i by

$$\begin{aligned} \mathbf{L}_i &= (x_{i-1} - x_i\,,\, y_{i-1} - y_i) \\ \mathbf{L}_{ij} &= (x_j - x_i\,,\, y_j - y_i) \end{aligned} \tag{8.4}$$

where $j = i-2,\, i-3,\, i-4, \ldots$. If ϕ_j denotes the angle between $\mathbf{L}_i$ and $\mathbf{L}_{ij}$ (Figure 8.1) then we propose to compute ϕ_j by the relation

$$\phi_j = \cos^{-1}\left(\frac{\mathbf{L}_i \,.\, \mathbf{L}_{ij}}{|\mathbf{L}_i|\,|\mathbf{L}_{ij}|}\right). \tag{8.5}$$

ϕ_j is computed giving decrement to j until for some j

$$\phi_{j+1} < \phi_j \leq \phi_{j-1}\,. \tag{8.6}$$

When the last relation holds for three consecutive values of j $(j+1,\, j,\, j-1)$ then the set of points $\{\, p_{i-1},\, p_{i-2}, \ldots, p_{j+1} \,\}$ is said to constitute the *left* arm of the point p_i and $l = i - j - 1$ is called the size of the left arm of p_i.
The region of support of p_i is then given by the set of points

$$\{\, p_{i-l}, \cdots, p_{i-1}, p_i, p_{i+1}, \cdots, p_{i+k} \,\}.$$

This technique for determination of arms is illustrated with the help of the figure-8 curve as shown in Figure 8.2. For the ith point as shown in the Figure 8.2, the vector $\mathbf{R}_i$ has the components (0,1) and the vector $\mathbf{R}_{ij} = \mathbf{R}_{i,i+2}$ has the components

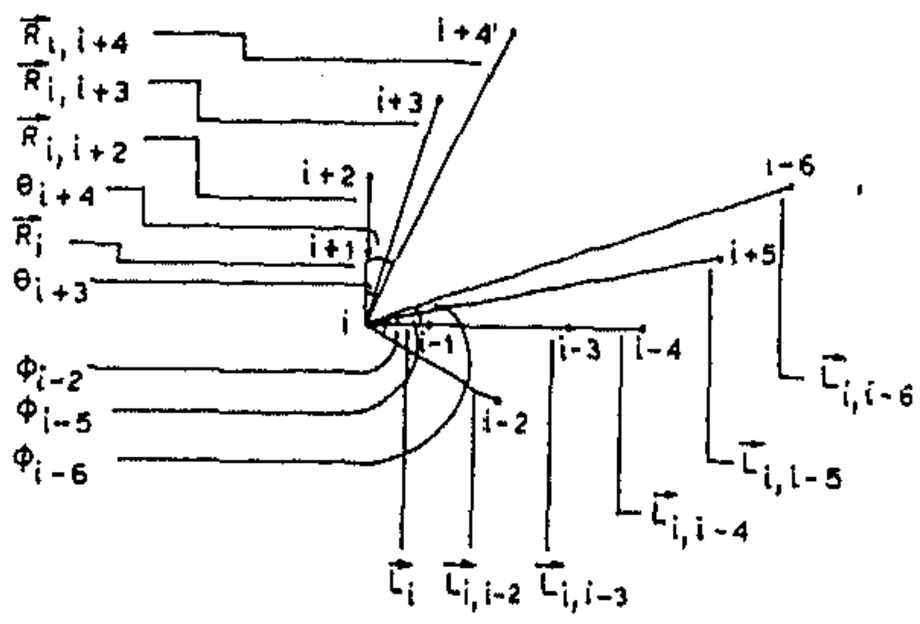


Figure 8.2: Illustrating computation of θ's and ϕ's

(0,2) and so using (8.2) we get $\theta_{i+2} = 0°$. Similarly $\theta_{i+3} = 18.4°$, $\theta_{i+4} = 26.6°$. And since $\theta_{i+2} < \theta_{i+3} < \theta_{i+4}$ hence the points p_{i+1} and p_{i+2} comprise the right arm of the point p_i and the length of the arm is $k = j - i - 1 = i + 3 - i - 1 = 2$.

Again for the ith point $\mathbf{L}_i$ has the components (1,0) and $\mathbf{L}_{ij} = \mathbf{L}_{i,i-2}$ has the components (2,-1) and hence using (8.5) we get $\phi_{i-2} = 26.6°$. Similarly $\phi_{i-3} = 0°$, $\phi_{i-4} = 0°$, $\phi_{i-5} = 11.3°$ $\phi_{i-6} = 18.6°$. And since $\phi_{i-4} < \phi_{i-5} < \phi_{i-6}$ hence the points $p_{i-1}, p_{i-2}, p_{i-3}, p_{i-4}$ comprise the left arm of the point p_i and $l = i - j - 1 = i - i + 5 - 1 = 4$ is the length of left arm.

The region of support of p_i is the set of points

$$\{ p_{i-4}, p_{i-3}, p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2} \}.$$

The region of support of the other points is computed in a similar manner.

8.3 Procedure: Determination of asymmetric region of support

Begin

Step 1. Define

$$\mathbf{R}_i = (x_{i+1} - x_i\,,\, y_{i+1} - y_i)$$
$$\mathbf{R}_{ij} = (x_j - x_i\,,\, y_j - y_i)$$

where $j = i+2,\, i+3,\, i+4, \ldots$.

Compute

$$\theta_j = cos^{-1}\left(\frac{\mathbf{R}_i \,.\, \mathbf{R}_{ij}}{|\mathbf{R}_i|\,|\mathbf{R}_{ij}|}\right)$$

giving increment to j until for some j

$$\theta_{j-1} < \theta_j \leq \theta_{j+1}\,.$$

$k = j - i - 1$ is the length of the right arm and the set of points $\{\, p_{i+1}, p_{i+2}, \ldots, p_{i+k} \,\}$ comprise the right arm of the point p_i .

Step 2. Define

$$\mathbf{L}_i = (x_{i-1} - x_i\,,\, y_{i-1} - y_i)$$
$$\mathbf{L}_{ij} = (x_j - x_i\,,\, y_j - y_i)$$

where $j = i-2,\, i-3,\, i-4, \ldots$.

Compute

$$\phi_j = cos^{-1}\left(\frac{\mathbf{L}_i \,.\, \mathbf{L}_{ij}}{|\mathbf{L}_i|\,|\mathbf{L}_{ij}|}\right)$$

giving decrement to j until for some j

$$\phi_{j+1} < \phi_j \leq \phi_{j-1}\,.$$

$l = i - j - 1$ is the length of the left arm and the set of points $\{\, p_{i-1}, p_{i-2}, \ldots, p_{i-l} \,\}$ comprise the left arm of the point p_i .

Step 3. The region of support of p_i is the set of points

$$\{\, p_{i-l}, \ldots, p_{i-1}, p_i, p_{i+1}, \ldots, p_{i+k} \,\}.$$

End.

8.4 Detection of dominant points

Once the right arm and the left arm of each point is determined by the last procedure, we define the right vector of p_i. The right vector is denoted by $\mathbf{a}_{ik}$ and is defined by

$$\mathbf{a}_{ik} = (x_{i+k} - x_i\,,\, y_{i+k} - y_i) \tag{8.7}$$

and the left vector at p_i is denoted by $\mathbf{b}_{il}$ and is defined by

$$\mathbf{b}_{il} = (x_{i-l} - x_i\,,\, y_{i-l} - y_i). \tag{8.8}$$

We define $k - l$ cosine at p_i by

$$\cos_{ikl} = \frac{\mathbf{a}_{ik} \,.\, \mathbf{b}_{il}}{|\,\mathbf{a}_{ik}\,|\,|\,\mathbf{b}_{il}\,|}\,. \tag{8.9}$$

A two-stage procedure is applied to detect dominant points. At the first stage, some input threshold is applied to the $k - l$ cosine to eliminate those points from consideration whose $k - l$ cosine is too small (≤ -0.800). At the second stage, a process of non-maxima suppression is applied to the remaining points to eliminate points whose $k - l$ cosine are not local maxima with respect to its immediate neighbors. The points remaining after these two stages are the dominant points. To improve the compression ratio, if two successive points of a curve appear as dominant points, the point with smaller region of support is suppressed.

The polygonal approximation is made by joining the adjacent dominant points.

8.5 Procedure: Detection of dominant points and polygonal approximation

Begin

Step 1. Define a k—vector at p_i as

$$\mathbf{a}_{ik} = (x_{i+k} - x_i\,,\, y_{i+k} - y_i)$$

and an l—vector at p_i as

$$\mathbf{b}_{il} = (x_{i-l} - x_i\,,\, y_{i-l} - y_i).$$

Step 2. Define $k-l$ cosine at p_i as

$$\cos_{ikl} = \frac{\mathbf{a}_{ik} \cdot \mathbf{b}_{il}}{|\mathbf{a}_{ik}|\,|\mathbf{b}_{il}|}.$$

Step 3. Suppress those points whose $k-l$ cosine ≤ -0.800.

Step 4. If $\cos_{ikl}$ lies between $\cos_{i-1,kl}$ and $\cos_{i+1,kl}$ then suppress p_i
else p_i is a dominant point.

Step 5. If two successive points of a curve appear as dominant points then the point with smaller region of support is suppressed.

Step 6. The points remaining after Step 5 are joined successively to make a polygonal approximation of the curve.

End.

8.6 Approximation errors

The accuracy of the location of dominant points and the closeness of the polygon to a digital curve can be determined by the pointwise error between the digital curve and its approximating polygon. We measure this error by the perpendicular distance of the point p_i's from their approximating line segment and denote it by e_i. Two error norms are defined as

1) Integral square error

$$E_2 = \sum_{i=1}^{n} {e_i}^2,$$

2) Maximum error

$$E_\infty = \max_i |e_i|.$$

8.7 Experimental results

We have applied the procedure developed in this chapter on the four digital curves used in the last chapter. The digital curves and their corresponding approximations are shown in Figures 8.3 through 8.6. In an attempt to focus on the efficiency of the procedure for detection of dominant points and polygonal approximation, we have

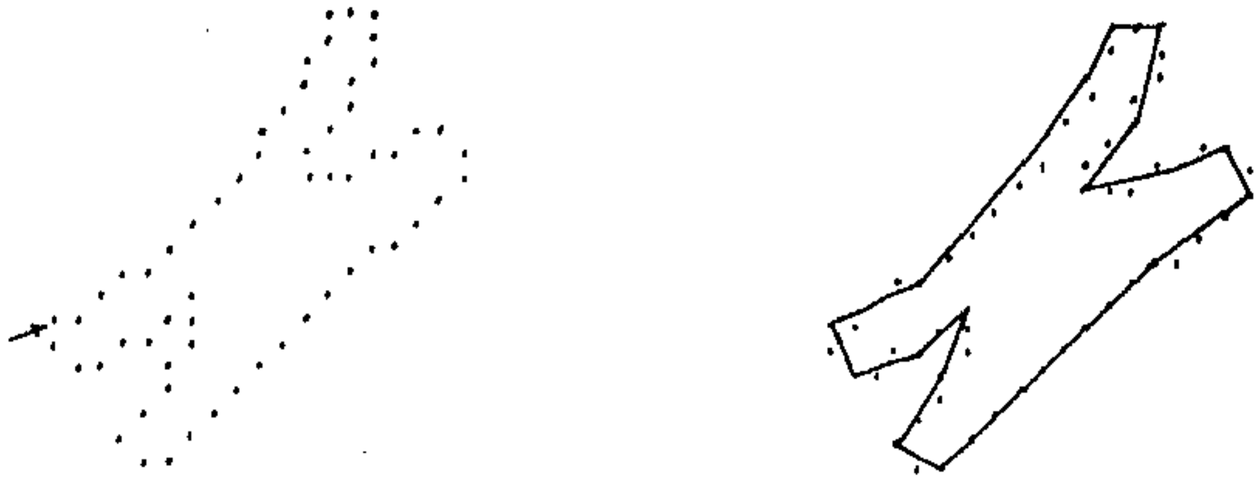


Figure 8.3: The chromosome shaped curve and its polygonal approximation.

Figure 8.4: The figure-8 curve and its polygonal approximation.

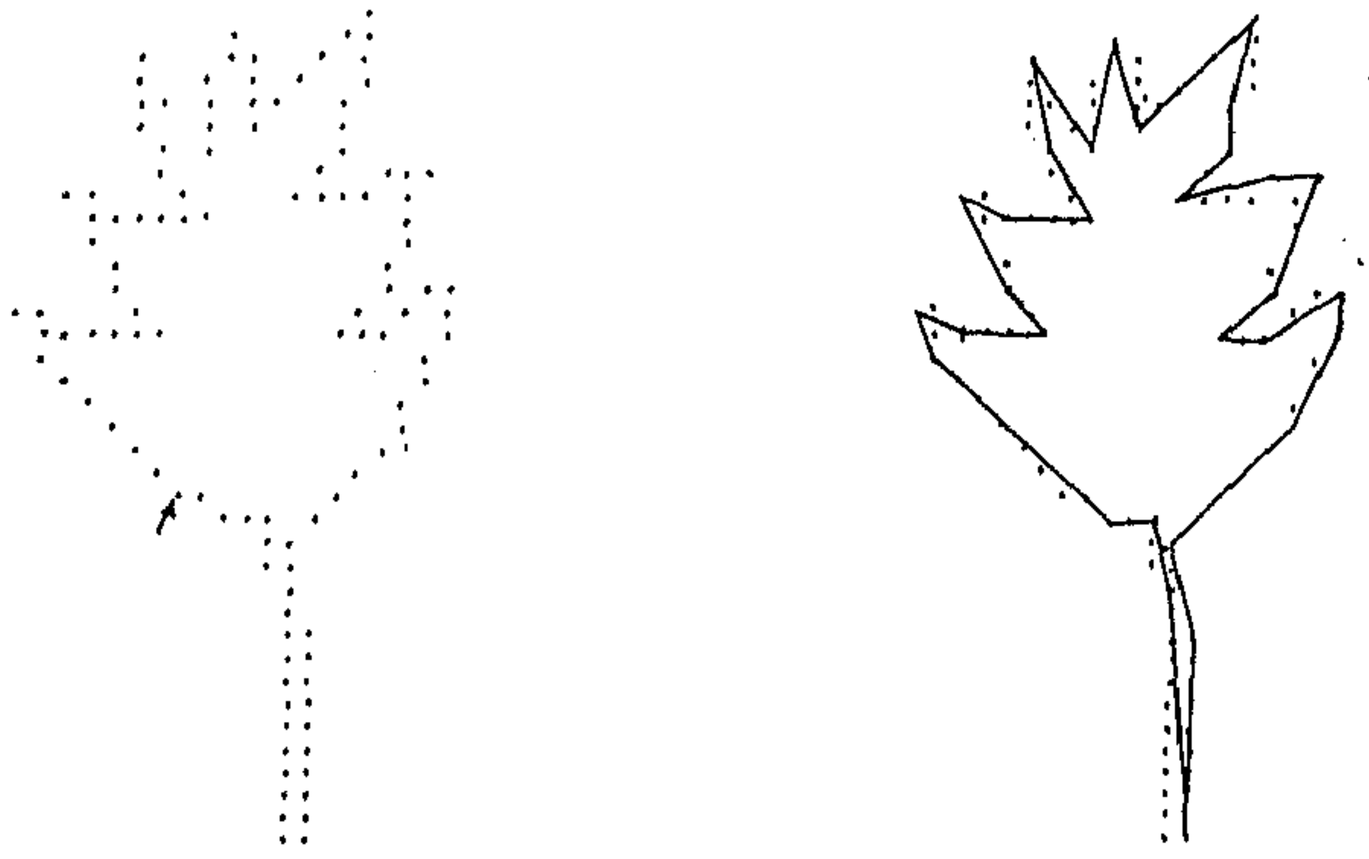


Figure 8.5: The leaf-shaped curve and its polygonal approximation.

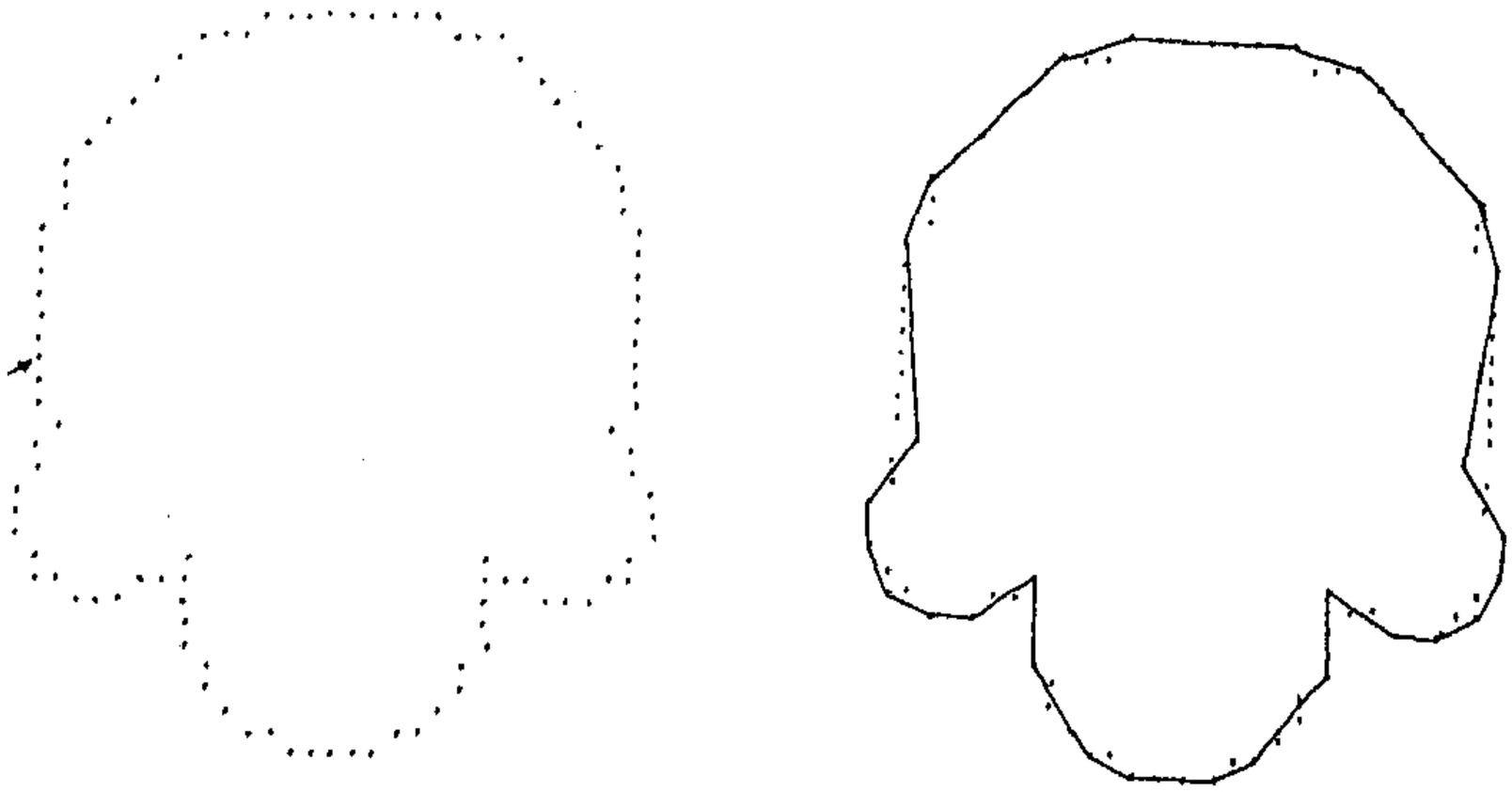


Figure 8.6: The curve with four semi-circles and its polygonal approximation.

computed the data compression ratio, the maximum error and the integral square. The results are displayed in the Table 8.1 which also contains the results of the Teh and Chin algorithm (reproduced from their work).

Table 8.1 A comparison between algorithm 8.5 & Teh-Chin algorithm

Digital curve	Chromosome	Figure-8	Leaf	Fig. 8.6
Number of points (n)	60	45	120	102
Results of algorithm 8.5				
Number of dominant points(n_d)	18	14	32	28
Compression ratio(n/n_d)	3.33	3.21	3.75	3.64
Integral square error	5.42	4.81	13.40	9.27
Maximum error	0.64	0.73	0.996	0.88
Results of Teh-Chin algorithm				
Number of dominant points(n_d)	15	13	29	22
Compression ratio(n/n_d)	4.00	3.50	4.10	4.60
Integral square error	7.20	5.93	14.96	20.61
Maximum error	0.74	1.00	0.99	1.00

8.8 Discussion

We have made no attempt of comparing this procedure with the existing ones except with the Teh and Chin algorithm. We find the following features of the procedure.
1) This procedure, unlike the existing ones, uses asymmetric region of support and asymmetric cosine.
2) This procedure, like that of Sankar-Sharma and Teh and Chin, need no parameter.
3) Sankar-Sharma's procedure need no input parameter but it does not determine region of support. The present procedure, like that of Teh and Chin does determine region of support.
4) Our procedure detects more dominant points than the Teh and Chin algorithm.
5) All processing are done locally, so it is be suitable for parallel processing.

Chapter 9

Scale-space analysis and corner detection on chain coded curves

As already stated in the opening chapter the fundamental problem in data smoothing is choice of input parameter. Too large a parameter will smooth out many important features (corners, vertices, zero-crossings) and too small a parameter will produce many redundant features. This is the fundamental problem of scale because, features appearing on a curve vary enormously in size and extent and there is seldom any basis of choosing a particular parameter for a feature of particular size. This problem may be resolved by automatic parameter tuning. The parameter size can be tuned on the basis of the local properties of the curve using a suitable criterion function. Teh and Chin [56] resolve this problem using chord length and perpendicular distance as the criterion function to determine the region of support of each point on the basis of the local properties of the curve. In chapter 7 and chapter 8 we too, have developed two such schemes in which automatic parameter tuning on the basis of the local properties of a curve is suggested.

Another approach to the solution of the problem of scale (choice of input parameter) is scale-space analysis.

In this chapter we propose to make scale-space analysis of digital curves using the polygonal approximation scheme presented in chapter 3 and show its application to corner detection [46]. Our scale parameter is discrete in nature. As the scale parameter is varied the location of the vertex points of the polygon is plotted against

the ordinal number of the points of the digital curve. The map showing the location of the vertex points over scales is called a scale-space map. This scale-space map is used to detect and locate corners on digital curves.

9.1 Scale-space map

The polygonal approximation discussed in chapter 3 involves a counter c which is responsible for imparting different degrees of smoothing to a curve. As the counter is varied different sets of vertex points are obtained. For small values of the counter the approximation results in a large number vertices and for large values of the counter the approximation results in a small number of vertices. For small values of c the vertices are located at fine levels of detail (low smoothing) and for large values of c the vertices are located at coarse levels of detail (high smoothing). As c is varied the curve is analyzed at different levels of detail. So c is called a scale parameter [7]. This scale parameter is discrete in nature.

As the values of c are increased from zero in a step size of one, different sets of vertex points result and different polygonal approximations are obtained. For $c = 0$ all points of a digital curve are vertex points. For $c = 1$ each point where the curve changes its direction is a vertex point. The number of vertex points for $c = 1$ is less than that for $c = 0$. As c is increased further the number of vertex points decreases resulting in higher smoothing, though two or more successive values of c may produce the same number of vertices.

The striking feature is that as c is varied from small values to large values, no new vertex point is introduced; but as c decreases from large values to small values, new vertex point may be introduced but the existing ones never disappear.

As a digital curve is smoothed out with different values of c, different polygonal approximations result each of which reflects a specific level of detail of the curve. These information are integrated in the form a scale-space map.

The $x-$axis (horizontal) indicates the ordinal number i of the point p_i of a digital curve and the $y-$axis (vertical) indicates the values of the counter c. As the counter is varied from small values to large values the position of the vertex points are plotted on the xy half plane. The map showing the position of the vertex points at

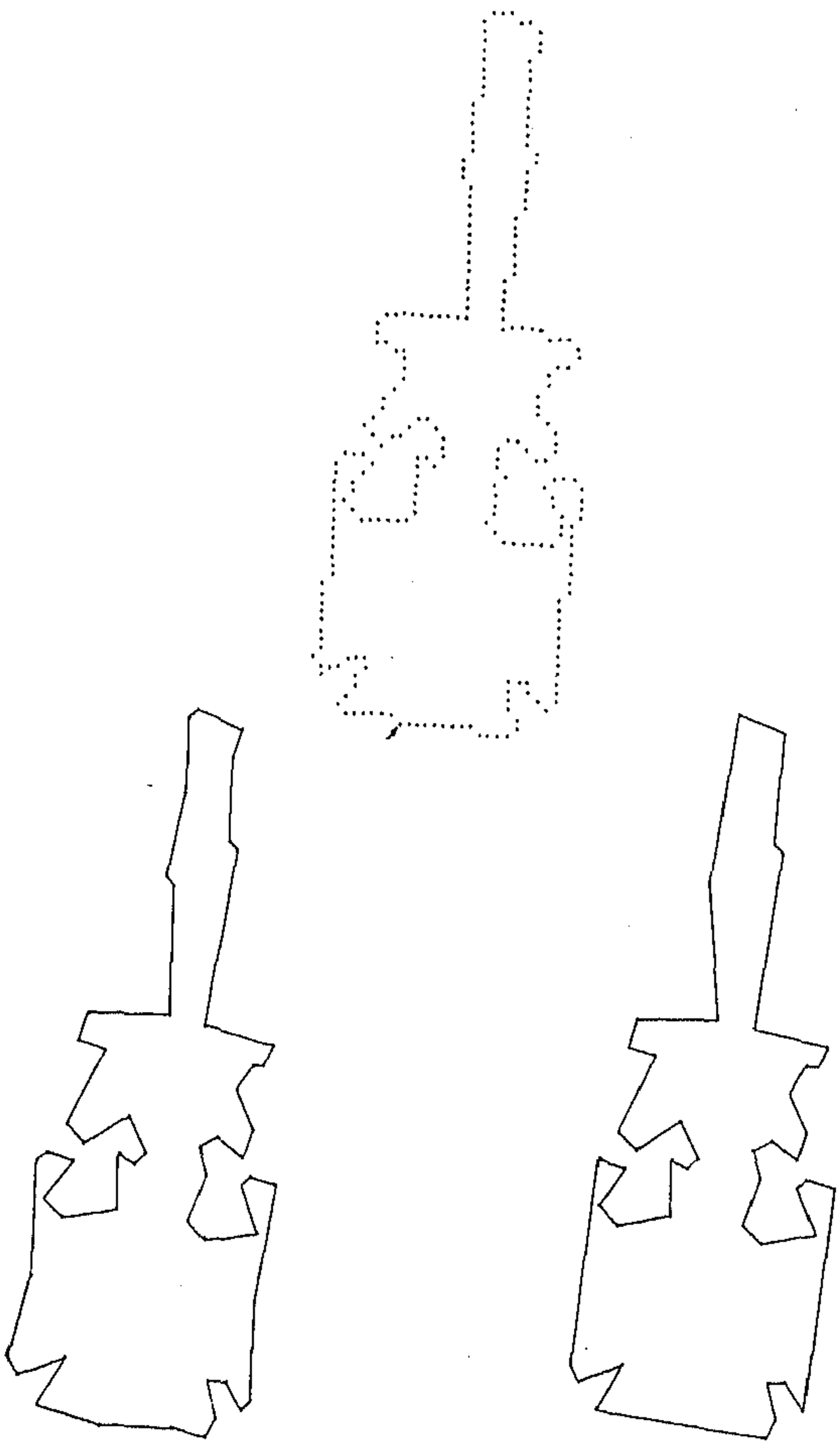
different values of the counter is called scale-space map. The map consists of a series of vertical lines at each vertex point. Some of these lines grow indefinitely and some other terminate as the counter increases. The vertical lines that grow indefinitely are indicative of those vertices which are detected at all levels of detail, fine as well as coarse. Whereas, those lines that terminate are indicative of those vertices that are detected at fine level of detail but disappear as the degree of smoothing increases. The location of the vertices at any scale is obtained by the orthogonal projection of the vertical lines on the $x-$axis.

We have constructed a scale-space map of the digital screwdriver. The digital curve and its polygonal approximations for different values of c are shown in Figure 9.1 and the scale-space map is shown in Figure 9.2. The counter has been varied from zero to 32 $(= 2^5)$. As the counter increases the number of vertex points decreases. We find that for $c \geq 15$ the number of vertices remains unaltered indicating the existence of a stable scale. The scale-space map is obtained by combining the different sets of vertex points that are obtained through the variation of c as already discussed. The curve is described in the clockwise direction starting from the point marked with an $\nearrow$ on the digital curve of Figure 9.1.

9.2 Scale-space behaviour of corner models and corner detection

In this section we are studying the scale-space behaviour of different corner models, namely, Γ model, END model and STAIR model as presented in [39]. We consider these models on digital domain where each model is represented by a sequence of integer coordinate points.

Figure 9.3 shows Γ models with different included angles and their respective scale-space map. As we are considering eight-connected curves, the included angles can only be a multiple of $\pi/4$. As seen from these figures, if the included angle be $3\pi/4$ then the scale-space map of the model consists of a single line pattern which vanishes ultimately as the counter increases. If the included angle be either $\pi/4$ or $\pi/2$ then the scale-space map consists of a persistent line pattern which never



(a) (b)

Figure 9.1: The screwdriver and its polygonal approximation for varying values of c, (a) $c = 8$, (b) $c \geq 15$.

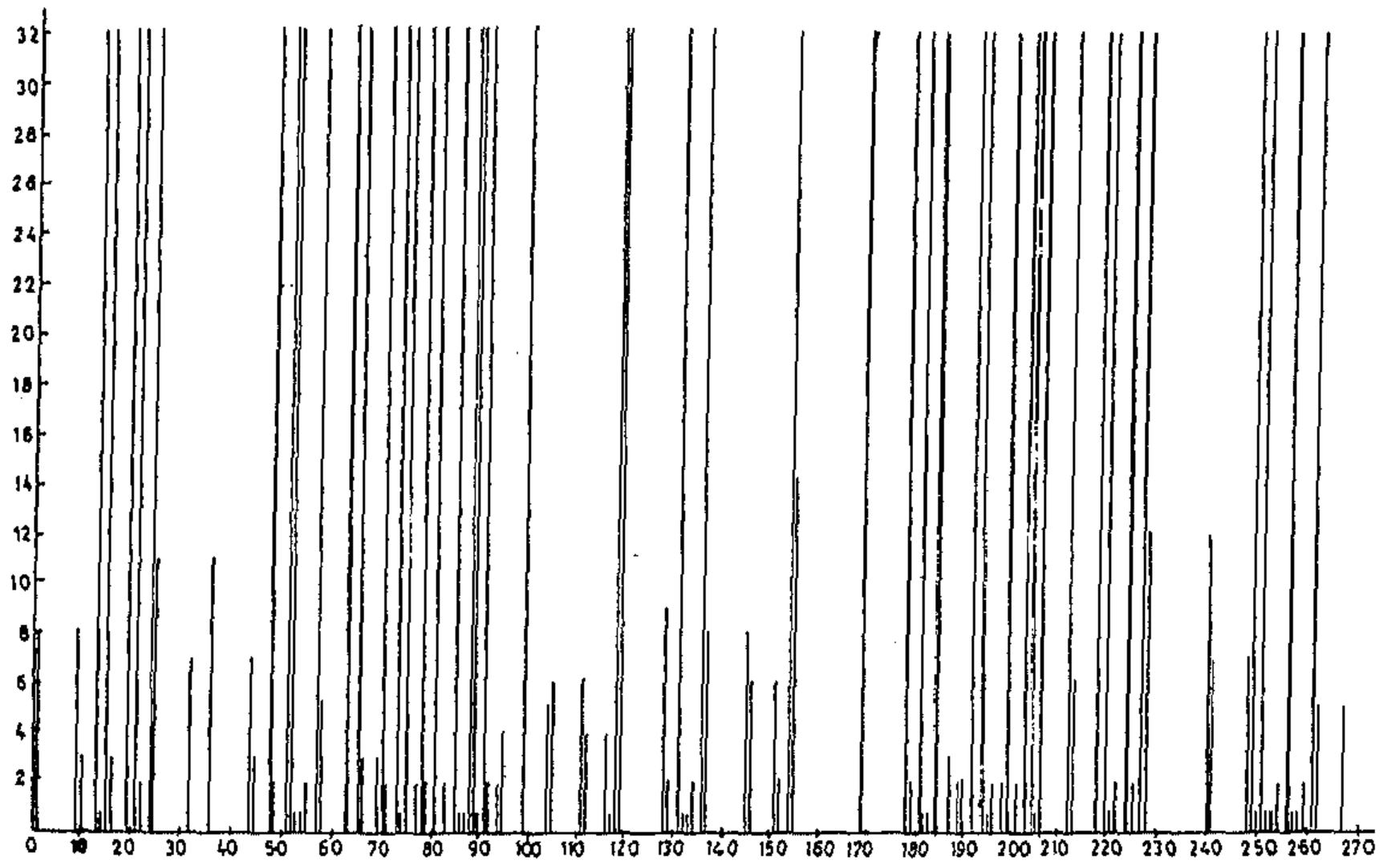


Figure 9.2: Scale-space map of the screwdriver.

disappears no matter how large the counter is.

Figure 9.4 shows END models with different included angles and their respective scale-space map. Here too, the included angle can only be multiple of $\pi/4$. As seen from these figures, if both the included angles be $3\pi/4$ then scale-space map consists of two line patterns one of which survives and the other disappears as the counter increases. If one angle be $3\pi/4$ and the other be either $\pi/4$ or $\pi/2$ then also the scale-space map exhibits the same behaviour except that in the former case (when both the included angles were $3\pi/4$) the line pattern which is farther from the starting point persists, whereas, in the latter case the line pattern that corresponds to the included angle not equal to $3\pi/4$ only persists. If none of the included angles be $3\pi/4$ then the scale-space map of the END model consists of two persistent line pattern none of which disappears no matter how large the counter is, showing that the END model is a combination of two Γ models with included angle other than $3\pi/4$.

Figure 9.5 shows STAIR models with different included angles (which can only be a multiple of $\pi/4$) and their respective scale-space map. As seen from these figures, if both the included angles be $3\pi/4$ then the scale-space map of the model consists of two line patterns both of which vanish as the counter increases. If the leading arm, the trailing arm and the base consist of equal number of points then both the line patterns disappear simultaneously, otherwise, one vanishes earlier than the other depending on the length of the arms and the base of the STAIR. If one of the included angles be $3\pi/4$ and the other be not equal to $3\pi/4$ then the scale-space map consists of two line patterns, but the line pattern corresponding to the angle of $3\pi/4$ disappears and the other persists as the counter increases indefinitely. If none of the included angles be $3\pi/4$ then the scale-space map consists of two persistent line patterns none of which disappears no matter how large the counter is, showing that the STAIR model with included angles other than $3\pi/4$ is a combination of two Γ models with included angle other than $3\pi/4$.

The central problem is how far the counter should be varied. One can observe that the maximum value of the counter cannot exceed the number of points of the digital curve. But it is not necessary to make the counter vary upto this limit. To find the maximum value of the counter we look for a series of adjoining STAIR

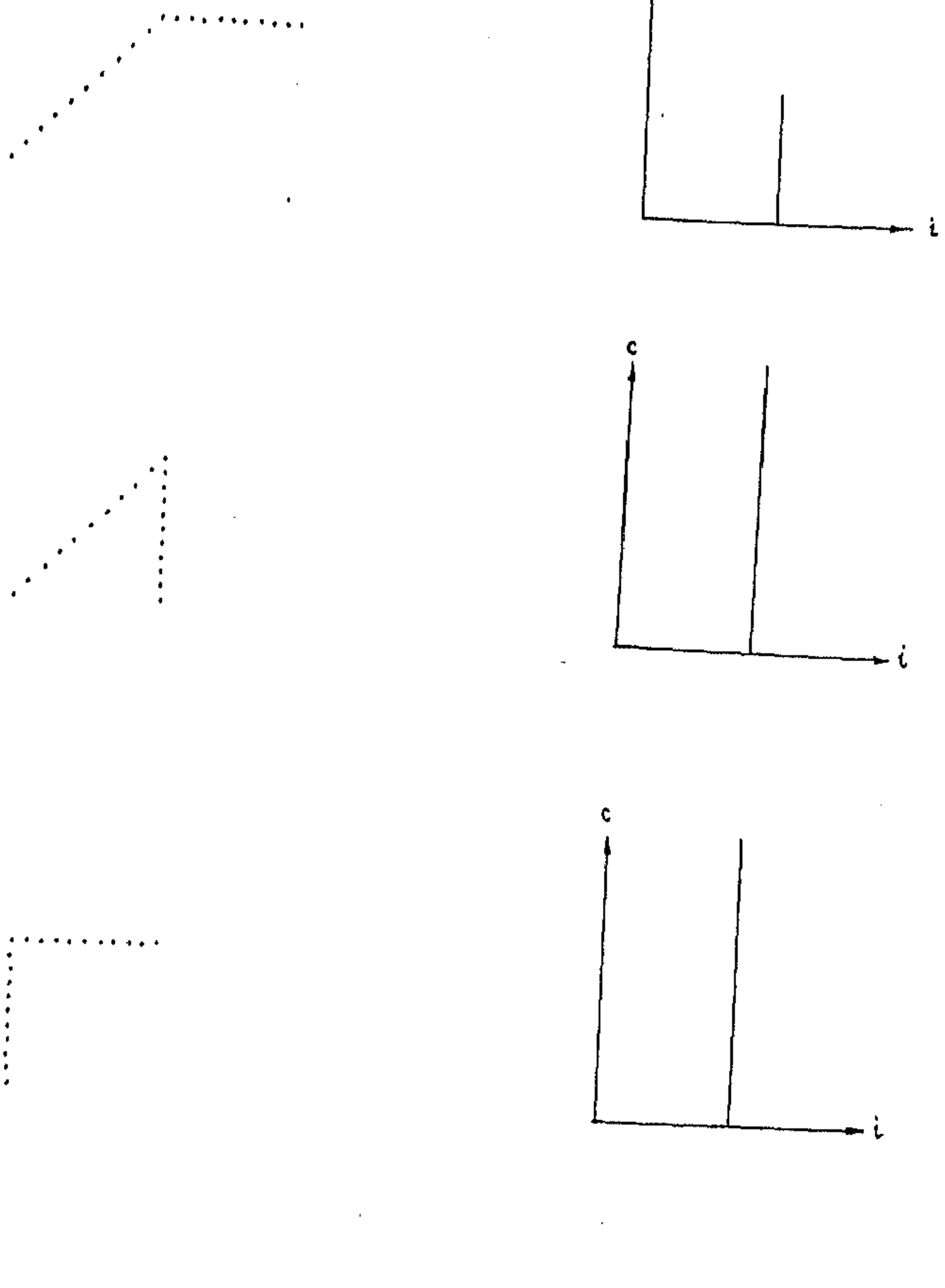


Figure 9.3: Γ models and their scale-space map. The left figure is a model and right is its scale-space map.

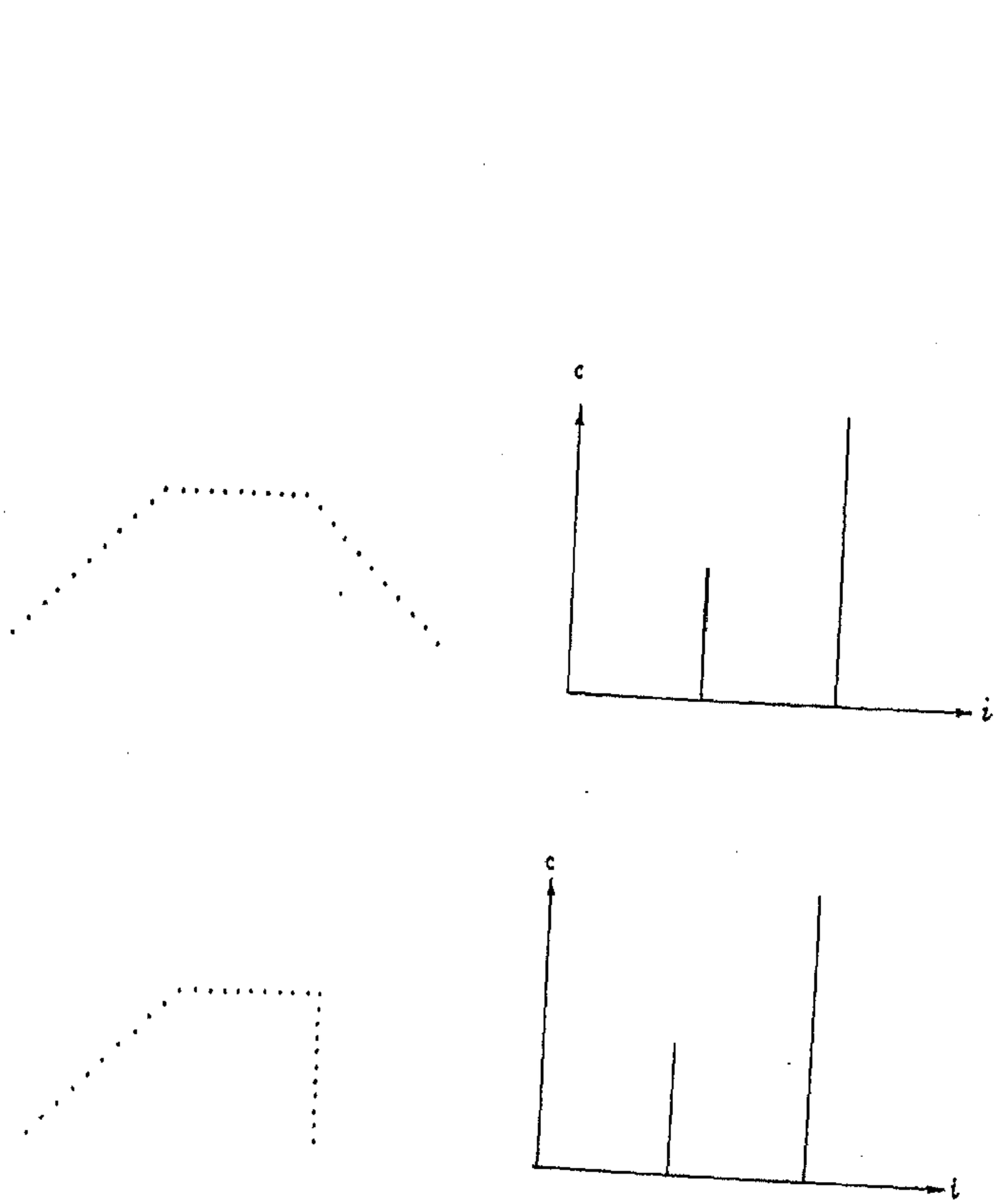
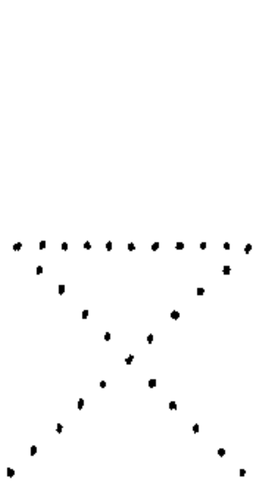
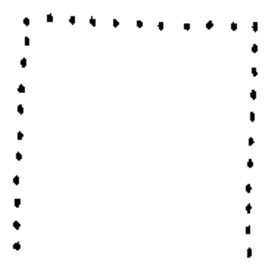


Figure 9.4: END models and their scale-space map. The left figure is a model and right is its scale-space map.





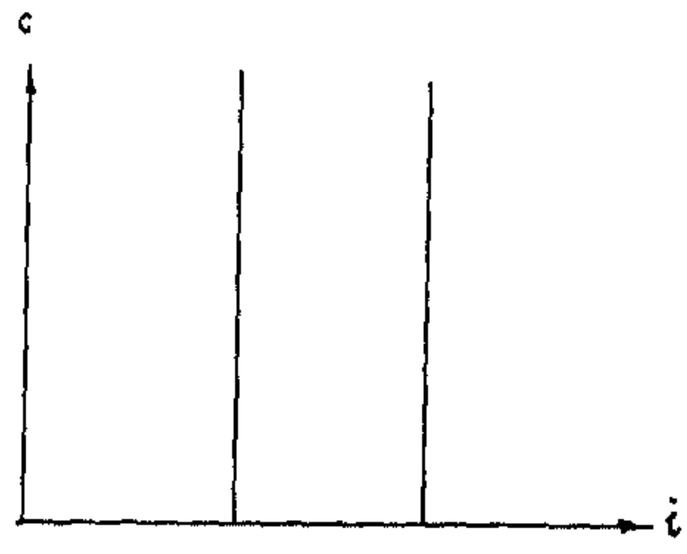


Figure 9.4: Continued.

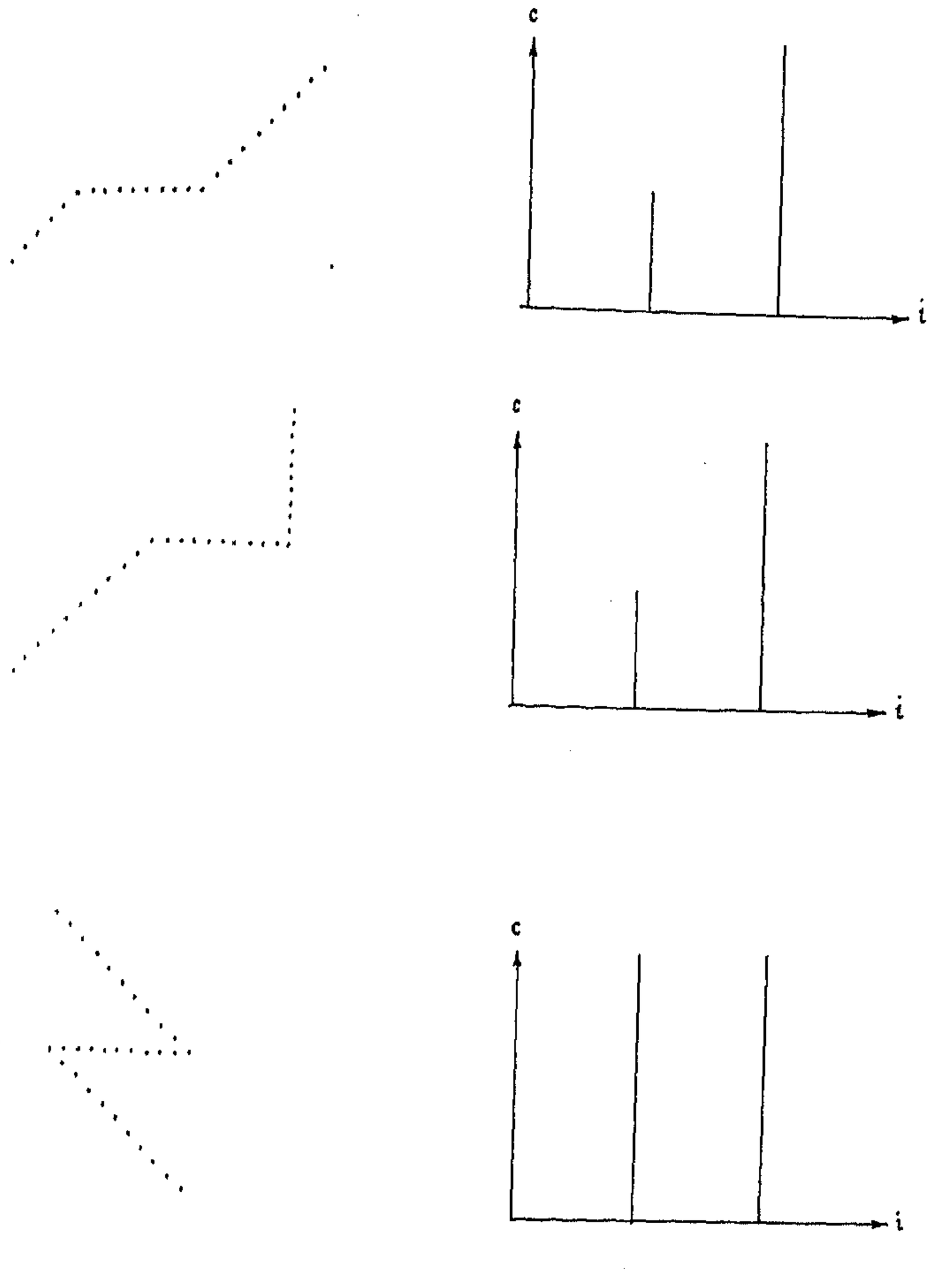
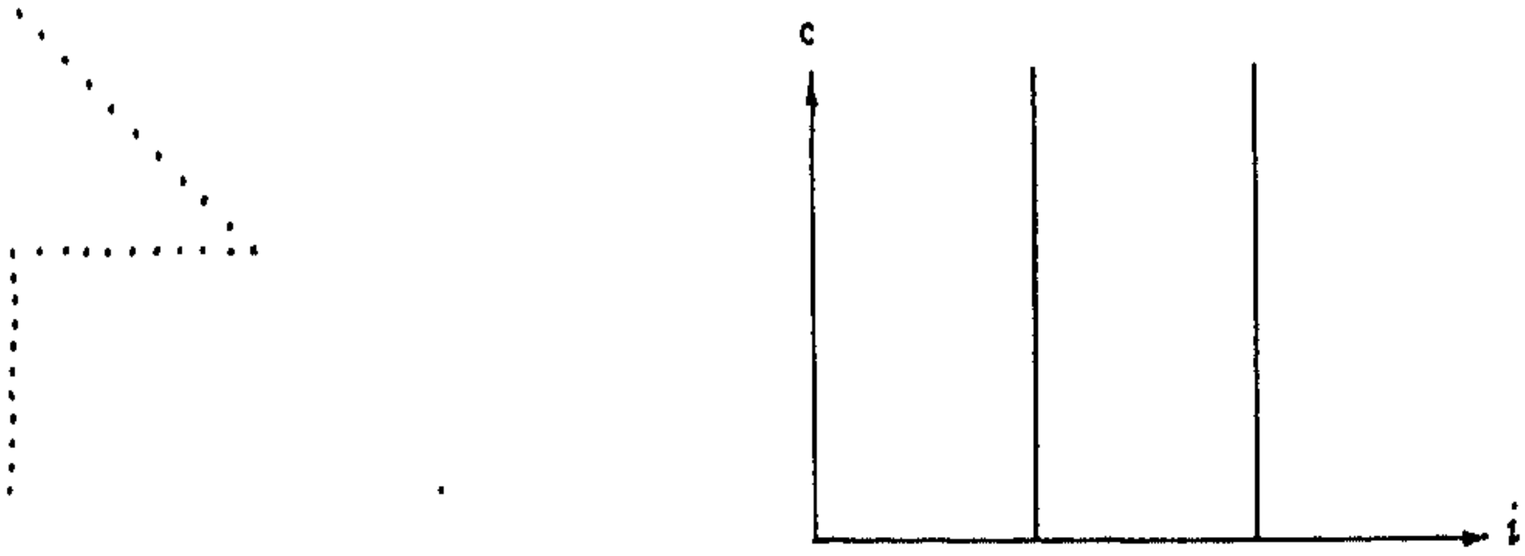


Figure 9.5: STAIR models and their scale-space map. The left figure is a *model* and the right is its scale-space map.



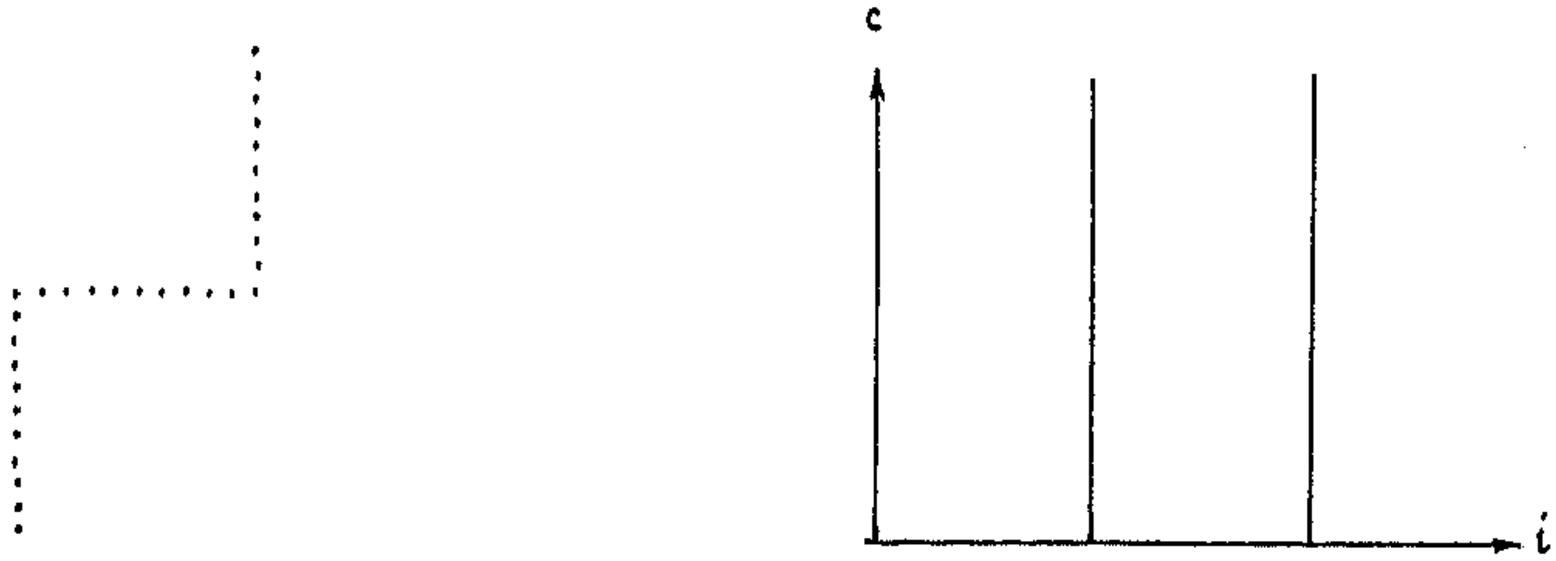


Figure 9.5: Continued.

models (of the digital curve) with included angles equal to $3\pi/4$. The maximum value of the counter can be taken to be equal to the total number of points in this longest series. This observation is in direct consequence of the scale-space behaviour of different corner models.

Following the above discussion we derive the following stability criteria for corner detection from the scale-space map of a digital curve:

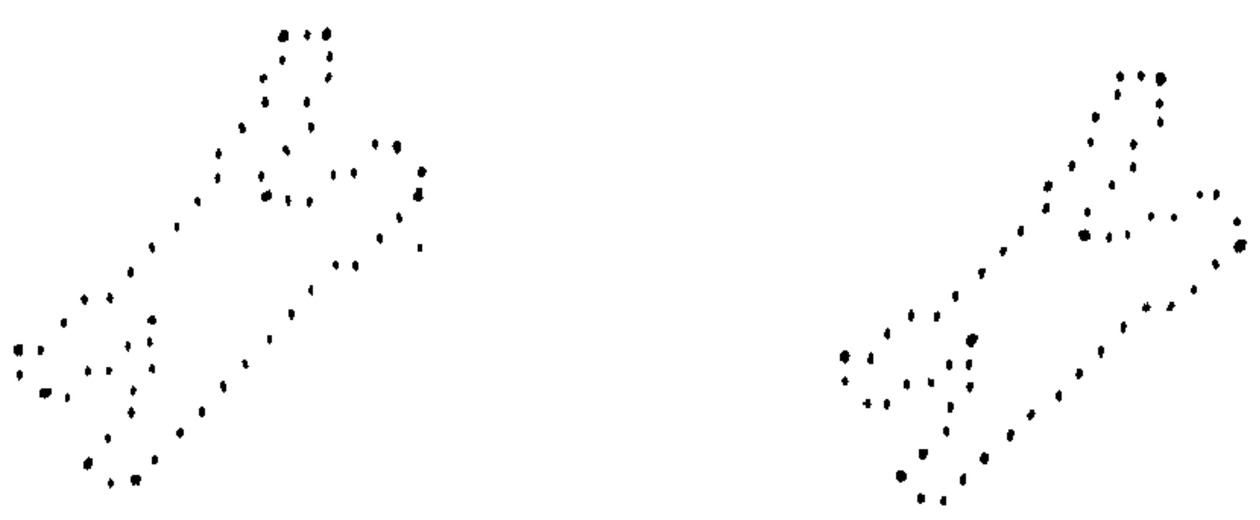
1) The lines that persist upto the maximum value of the counter correspond to corner points.

2) The lines that do not survive all scales (all values of the counter upto its maximum) but in the immediate neighborhood of the persistent lines also correspond to corner points provided these non-surviving lines are separated from the persistent lines by more than unit length in the scale-space map.

3) The pair of lines that do not survive all scales correspond to corner points provided that they are separated by more than unit length in the scale-space map.

These stability criteria are the direct consequence of the scale-space behaviour of different corner models.

In order to detect corners from a digital curve, a scale-space map of the digital curve is constructed and the corners are detected and located by analyzing the scale-space map with the help of the stability criteria. The corners that are detected by this process are not necessarily true corners, some of them may be redundant due to quantization noise and boundary noise. In order to remove these redundant corners we introduce a two-stage cleaning process. At the first stage, if two points p_i and p_{i+1} of a digital curve appear as corner points then the point p_i is retained and p_{i+1} is discarded. At the second stage, at each corner point we find out the perpendicular distance of the corner point from the line joining the two adjacent corner points. If this distance exceeds unity then the corner point is retained, otherwise it is discarded. This threshold unity is selected based on the fact that a slanted straight line is quantized into a set of either horizontal or vertical line segments separated by one pixel steps. In addition, we assume that the boundary noise are no more than one pixel, and if the noise level is known *a priori* then this threshold can be adjusted accordingly [39].

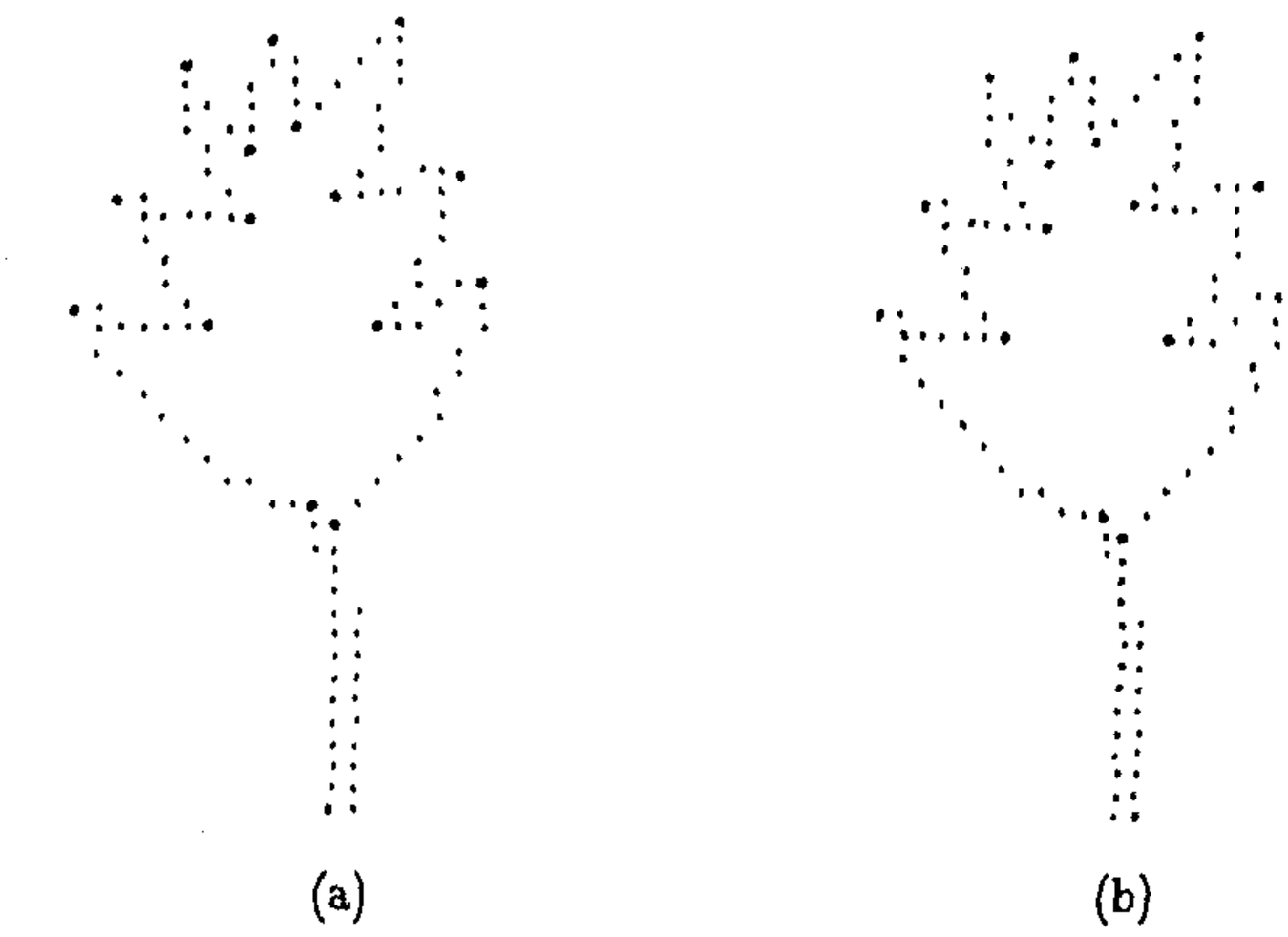


(a) (b)

Figure 9.6: The chromosome shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

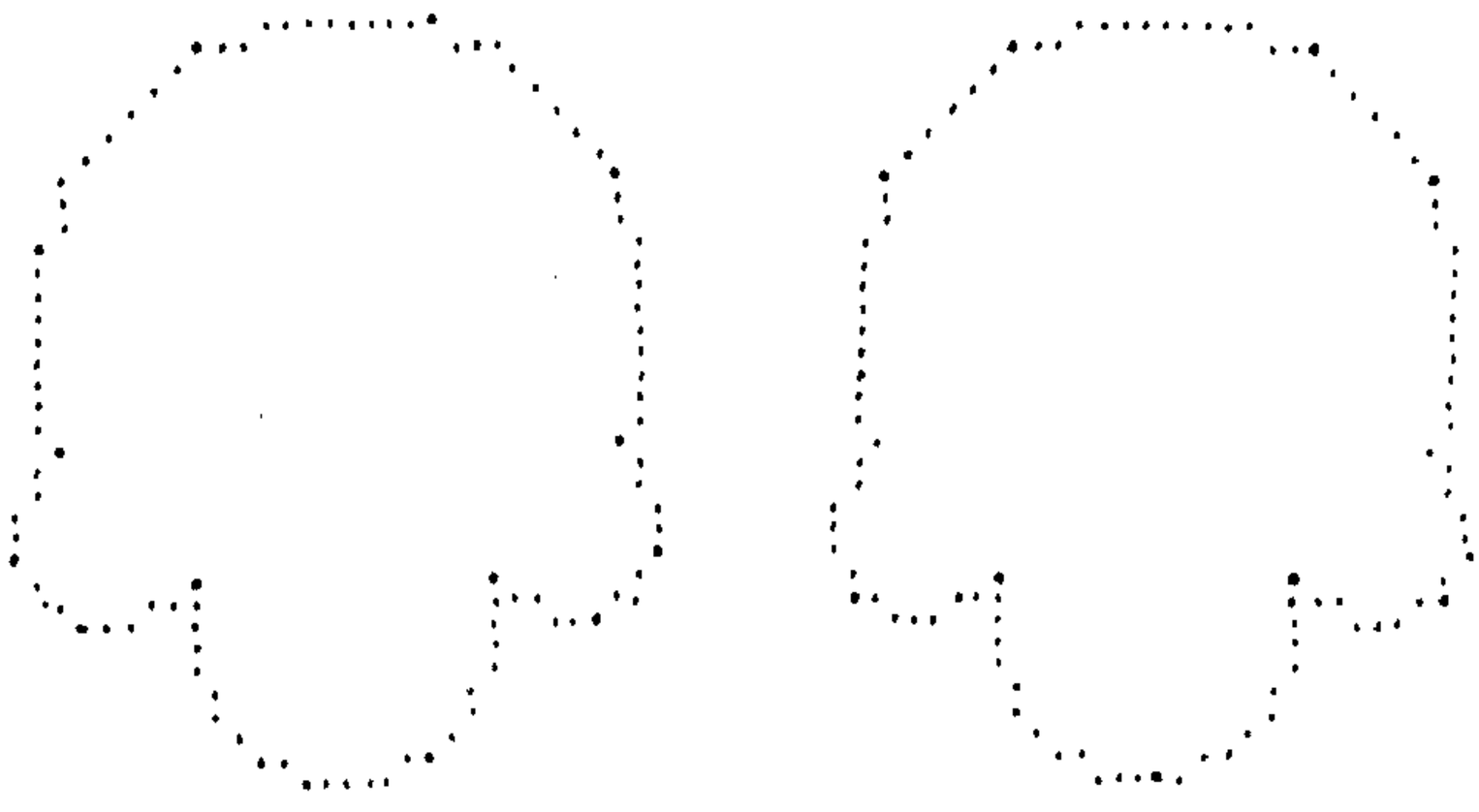
(a) (b)

Figure 9.7: The figure-8 curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.



(a) (b)

Figure 9.8: The leaf-shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.



(a) (b)

Figure 9.9: The curve with four semi-circles. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

9.3 Experimental results

The corner detector developed in this chapter is applied on four digital curves as shown in Figures 9.6 through 9.9. The corners are indicated by bold solid circles. These figures also show the results obtained by the Rattarangsi-Chin algorithm [39]. We find that our corner detector detects more corner points than the Rattarangsi-Chin algorithm.

Chapter 10

Scale-space analysis and corner detection using iterative Gaussian smoothing with constant window size

The scale-space analysis and corner detection scheme presented in the last chapter holds for digital curves with uniformly spaced points only. The scale-space analysis does not involve convolution of a curve with a smoothing kernel. Corner detection is done without estimating curvature. In this chapter we propose another scale-space analysis technique followed by corner detection. The procedure involves convolution of a digital curve with a smoothing kernel. Corner detection is done via curvature estimation. The scale-space analysis and corner detection scheme holds for digital curves with uniformly as well as non-uniformly spaced points.

Rattarangsi and Chin [39] make scale-space analysis using Gaussian kernel with varying window size. So the space requirements for the Gaussian filter coefficients is of the order of the square of data size. In this chapter we present an alternative approach to scale-space analysis followed by corner detection using iterative Gaussian smoothing with constant window size. As the window size is held constant, the space requirements for the Gaussian filter coefficients is finite and independent

of data size. In the following sections we present the iterative Gaussian smoothing process, show its convergence and determine the maximum number of iterations that can be performed on a closed digital curve without wrap around effects. A scale-space map showing the location of the maxima of absolute curvature at varying iteration scale is proposed. The map is shown to enjoy scale-space property. An analysis of the scale-space behavior of corner models is presented. The scale-space map of a digital curve is converted into a tree representation and corners are detected and located in a process of interpreting the tree. The space requirements and the computational load is discussed and compared with the most recent work [39]. Experimental results are presented to show the performance of the corner detector.

10.1 Gaussian smoothing and curvature measurement

For a continuous and smooth curve, the curvature at a point is defined as the rate change of tangential angle (ψ) with respect to the arc length (s). So the curvature at an arc length of s is given by

$$\kappa = \frac{d\psi}{ds}. \tag{10.1}$$

In the Cartesian coordinate system, if the equation of the curve is described by $y = f(x)$, the curvature at a point (x, y) is given by

$$\kappa(x) = \frac{\frac{d^2y}{dx^2}}{\left\{1 + \left(\frac{dy}{dx}\right)^2\right\}^{\frac{3}{2}}}. \tag{10.2}$$

If the equation of the curve is expressed parametrically with the arc length s as the parameter so that the curve is described by $x = x(s)$ and $y = y(s)$ then the curvature at a point s is given by

$$\kappa(s) = \frac{\frac{d^2y}{ds^2}\frac{dx}{ds} - \frac{d^2x}{ds^2}\frac{dy}{ds}}{\left\{\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2\right\}^{\frac{3}{2}}}. \tag{10.3}$$

But

$$\frac{dx}{ds} = \cos\psi \quad and \quad \frac{dy}{ds} = \sin\psi,$$

so

$$\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2 = 1$$

and

$$\kappa(s) = \frac{d^2y}{ds^2}\frac{dx}{ds} - \frac{d^2x}{ds^2}\frac{dy}{ds}. \tag{10.4}$$

The curve $(x(s), y(s))$ that arise in computer vision and other allied problems may be continuous but not smooth (a digital curve is neither continuous nor smooth). So smoothing is, in general necessary before we detect the significant local events of a curve. Smoothing is usually done with the Gaussian kernel as it has some attractive properties [7], [64]. To smooth a curve $(x(s), y(s))$ it is convolved with the Gaussian kernel

$$g(s, \sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-s^2/2\sigma^2}, \quad -\infty < s < +\infty. \tag{10.5}$$

We assume that the curve in question is closed with arc length S so that $0 \leq s \leq S$ and the parameter σ is sufficiently small so that three times σ does not exceed $S/2$ so as to avoid aliasing effects. The convolution of $(x(s), y(s))$ with the Gaussian kernel (10.5) is defined by

$$\begin{aligned} X(s, \sigma) &= x(s) * g(s, \sigma) \\ &= \int_{u=s-S/2}^{s+S/2} x(u)\, g(s-u, \sigma)\, du, \end{aligned} \tag{10.6}$$

$$\begin{aligned} Y(s, \sigma) &= y(s) * g(s, \sigma) \\ &= \int_{u=s-S/2}^{s+S/2} y(u)\, g(s-u, \sigma)\, du. \end{aligned} \tag{10.7}$$

As $g(s, \sigma)$ is maximally often differentiable and the convolution and differentiation are commutative so the derivatives of $X(s, \sigma)$ and $Y(s, \sigma)$ with respect to s do exist and in particular, their first and second derivatives are given by

$$\begin{aligned} \frac{dX}{ds} &= x(s) * \frac{dg(s, \sigma)}{ds}, \\ \frac{dY}{ds} &= y(s) * \frac{dg(s, \sigma)}{ds}, \end{aligned} \tag{10.8}$$

$$\frac{d^2X}{ds^2} = x(s) * \frac{d^2g(s,\sigma)}{ds^2},$$
$$\frac{d^2Y}{ds^2} = y(s) * \frac{d^2g(s,\sigma)}{ds^2}. \tag{10.9}$$

And the curvature of the Gaussian smoothed curve is given by

$$\kappa(s,\sigma) = \frac{d^2Y}{ds^2}\frac{dX}{ds} - \frac{d^2X}{ds^2}\frac{dY}{ds}. \tag{10.10}$$

Rattarangsi and Chin [39] find the first and second order derivatives of $\kappa(s,\sigma)$ with respect to s to locate the extreme curvature points for varying σ. Since the Gaussian kernel has negligible contribution beyond the 3σ limits hence the maximum value upto which the parameter σ is to be varied is determined by $3\sigma_{max} = S/2$ which leads to $\sigma_{max} = S/6$ so as to avoid aliasing effects [39].

10.2 Iterative Gaussian smoothing

To convolve a digital curve with the Gaussian kernel, Rattarangsi and Chin [39] use the digital Gaussian filter coefficients of [10] namely, $c_{-1} = 0.2236$, $c_0 = 0.5477$ and $c_1 = 0.2236$ for window size $w = 3$. These filter coefficients have been mentioned in [29] and [11] as the best approximation of the Gaussian distribution for $w = 3$. The digital Gaussian filter coefficients for window size higher than $w = 3$ are obtained by repeated convolutions of these coefficients with themselves. The maximum window size should not exceed the length of the curve so as to avoid the wrap around effects. Each window size corresponds to a specific value of the parameter σ of the Gaussian distribution. The greater the window size is, the higher is the value of the parameter.

In this chapter we make an alternative approach to the problem. Instead of smoothing the curve with varying window size, we repeatedly convolve the curve keeping the window size constant at $w = 3$. This approach has an advantage over that presented in [39]. In the later approach as the varying window size is used, the space requirements by the Gaussian filter coefficients is of the order of square of data size. But in the present approach the space requirements by the filter coefficients is finite, small and does not depend on data size. Moreover, it is shown later that

the computational load of the smoothing process in the Rattarangsi and Chin [39] algorithm is $O(n^2)$ whereas in the present approach it is $O(n)$.

The iterative convolution is performed with the digital Gaussian filter coefficients $c_{-1} = 0.2236$, $c_0 = 0.5477$ and $c_1 = 0.2236$ using the iterative process

$$\begin{aligned} X_i^t &= \sum_{m=-1}^{1} X_{i+m}^{t-1} c_m \\ Y_i^t &= \sum_{m=-1}^{1} Y_{i+m}^{t-1} c_m . \end{aligned} \tag{10.11}$$

(X_i^t, Y_i^t) denote the Gaussian smoothed coordinates of the ith point (x_i, y_i) at the tth iteration, $t = 1, 2, 3, ...$; and $X_i^0 = x_i$, $Y_i^0 = y_i$, $i = 1, 2, 3, ..., n$.

Though Saint-Marc et al. [52] use repeated weighted averaging process but they do not use the Gaussian filter coefficients. The kernel they use depends on input data. The kernel involves gradient of the signal. As gradient is orientation dependent so the smoothing process too, will depend on the orientation of the signal. A particular point of a signal will be subjected to different degree of smoothing for different orientation of the signal. But it is essential that the extent of smoothing should not depend on the orientation of the signal. On the other hand the Gaussian kernel is shift invariant (does not depend on data). The degree of smoothing subjected to a curve / signal depends only on the Gaussian filter coefficients.

10.3 Convergence

The repeated convolution of a digital curve with the digital Gaussian filter coefficients can be written in the matrix form as

$$\mathbf{X}^t = \mathbf{A}\mathbf{X}^{t-1} \quad \textit{and} \quad \mathbf{Y}^t = \mathbf{A}\mathbf{Y}^{t-1} \tag{10.12}$$

where **A** is the convolution matrix of order $(n+2) \times (n+2)$ given by

$$\mathbf{A} = \begin{bmatrix} c_0 & c_1 & & & & \\ c_{-1} & c_0 & c_1 & & & \\ & c_{-1} & c_0 & c_1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & c_{-1} & c_0 & c_1 \\ & & & & c_{-1} & c_0 \end{bmatrix}$$

and

$$\mathbf{X}^t = [X_n^t, X_1^t, X_2^t, ..., X_n^t, X_1^t],$$

$$\mathbf{Y}^t = [Y_n^t, Y_1^t, Y_2^t, ..., Y_n^t, Y_1^t]$$

are the column matrices of order $n+2$. The iterative process will converge if for some norm, **A** satisfies $\|\mathbf{A}\| \leq 1$. Since the infinity norm of a matrix is simply the maximum sum of the moduli of the elements of the matrix, hence the infinity norm of the convolution matrix **A** is $\|\mathbf{A}\|_\infty = c_{-1} + c_0 + c_1 < 1$. Again, the largest of the moduli of the eigen values of a square matrix cannot exceed its infinity norm i.e. $\rho(\mathbf{A}) \leq \|\mathbf{A}\|_\infty$. So the $2-norm$ of the matrix **A** is also bounded by unity. So the convolution process converges as $\|\mathbf{A}\| < 1$ [52].

10.4 Maximum number of iteration

For a closed digital curve with n points the number of iterations is determined by the data size n in order to avoid the aliasing effects. If the curve has an odd number of points (n odd) then the maximum number of iterations that can be performed is $(n-1)/2$ and if n be even then the maximum number of iterations that can be performed is $n/2 - 1$.

To show this we decompose the Gaussian smoothed coordinates (X_i^t, Y_i^t) in terms of (x_i, y_i). For this analysis we propose to write $c_{-1} = c_{-1}^1$, $c_0 = c_0^1$ and $c_1 = c_1^1$. So

$$X_i^t = \sum_{m=-1}^{1} X_{i+m}^{t-1} c_m^1$$

$$Y_i^t = \sum_{m=-1}^{1} Y_{i+m}^{t-1} c_m^1 .$$

But

$$X_i^{t-1} = \sum_{m=-1}^{1} X_{i+m}^{t-2} c_m^1$$

so

$$X_i^t = \sum_{m=-2}^{2} X_{i+m}^{t-2} c_m^2$$

where

$$c_{-2}^2 = c_{-1}^1,\ c_{-1}^2 = 2c_0^1 c_{-1}^1,\ c_0^2 = c_{-1}^1 + c_0^1 + c_1^1,\ c_1^2 = 2c_0^1 c_1^1,\ c_2^2 = c_1^1 .$$

Decomposing X_i^{t-2} in terms of X_i^{t-3} we get

$$X_i^t = \sum_{m=-3}^{3} X_{i+m}^{t-3} c_m^3 ,$$

c_m^3 is a homogeneous function of c_{-1}, c_0 and c_1 of degree 3, for each value of $m = -3, -2, -1, 0, 1, 2, 3$. So using mathematical induction, at the lth decomposition we get,

$$X_i^t = \sum_{m=-l}^{l} X_{i+m}^{t-l} c_m^l ,$$

where c_m^l is a homogeneous function of c_{-1}, c_0 and c_1 of degree l. So X_i^t in terms of x_i is

$$X_i^t = \sum_{m=-t}^{t} x_{i+m} c_m^t , \qquad t = 1, 2, \ldots$$

where c_m^t is a homogeneous function of c_{-1}, c_0, c_1 of degree t. Similarly,

$$Y_i^t = \sum_{m=-t}^{t} y_{i+m} c_m^t , \qquad t = 1, 2, \ldots$$

This result shows that at any iteration t the iterative convolution process actually takes into account the effects of the points $i-t$, $i-t+1, \ldots, i-1$, i, $i+1, \ldots$, $i+t-1$, $i+t$. So in order that a particular point of a curve is not evaluated twice during the convolution process i.e. to avoid the aliasing effects, none of the member of the set

$\{i-t, i-t+1, ..., i-1\}$ should be a member of the set $\{i+1, i+2, ..., i+t-1, i+t\}$ and vice-versa. This condition is satisfied if and only if

$$\begin{aligned} t_{max} &= n/2-1, \quad n \text{ even} \\ &= (n-1)/2, \quad n \text{ odd.} \end{aligned} \tag{10.13}$$

So the maximum number of iterations that can be performed on a closed digital curve with n points is

$$\begin{aligned} t_{max} &= n/2-1, \quad n \text{ even} \\ &= (n-1)/2, \quad n \text{ odd.} \end{aligned} \tag{10.14}$$

Though Saint-Marc et al. [52] has shown the convergence (which is too slow) of their iterative process but they could not suggest the number of iterations that should be performed on a closed digital curve.

For open digital curves the number of iterations to be performed should be so chosen that the end effects do not come into play. This can be done by choosing t_{max} properly so that it does not exceed the number of points on the right / left of any point being convolved. One can as well take the open curve to be sufficiently large so that one can perform a large number of iterations.

10.5 Curvature estimation and scale-space map

To compute the curvature at a point of a digital curve Rattarangsi and Chin [39] define the first and second order finite differences of (x_i, y_i) as

$$\Delta x_i = \frac{x_{i+1} - x_{i-1}}{\sqrt{(x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2}}, \tag{10.15}$$

$$\Delta y_i = \frac{y_{i+1} - y_{i-1}}{\sqrt{(x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2}}, \tag{10.16}$$

$$\Delta^2 x_i = \frac{\frac{x_{i+1}-x_i}{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}} - \frac{x_i-x_{i-1}}{\sqrt{(x_i-x_{i-1})^2+(y_i-y_{i-1})^2}}}{\frac{1}{2}\sqrt{(x_{i+1} - x_{i-1})^2 + (y_{i+1} - y_{i-1})^2}}, \tag{10.17}$$

$$\Delta^2 y_i = \frac{\frac{y_{i+1}-y_i}{\sqrt{(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2}} - \frac{y_i-y_{i-1}}{\sqrt{(x_i-x_{i-1})^2+(y_i-y_{i-1})^2}}}{\frac{1}{2}\sqrt{(x_{i+1}-x_{i-1})^2+(y_{i+1}-y_{i-1})^2}} \tag{10.18}$$

and the curvature measure is given by

$$\kappa = \Delta x\, \Delta^2 y - \Delta y\, \Delta^2 x. \tag{10.19}$$

We use these expressions for Δx, Δy, $\Delta^2 x$, $\Delta^2 y$ and κ with (x, y) being replaced by the corresponding Gaussian smoothed coordinates (X, Y). So the curvature measure of the Gaussian smoothed curve at the tth iteration and at the ith point is given by

$$\kappa_i^t = \Delta X_i^t\, \Delta^2 Y_i^t - \Delta Y_i^t\, \Delta^2 X_i^t. \tag{10.20}$$

At each iteration $t = 1, 2, 3, \ldots$ the local maxima of the absolute curvature $|\kappa_i^t|$ are detected. For a fixed t, the value i for which $|\kappa_i^t|$ exceeds $|\kappa_{i-1}^t|$ and $|\kappa_{i+1}^t|$ gives the location of a local maximum of the absolute curvature. As the iteration proceeds different sets of absolute curvature maxima for different values of t are obtained. These information are integrated in the form of a scale-space map of the digital curve. Along the $x-$axis (horizontal) the ordinal number i of the points is shown, along the $y-$axis (vertical) the number of iterations t is shown. As the iteration proceeds location of the maxima of absolute curvature are plotted on the xy half plane. The dot diagram showing the location of the absolute curvature maxima at different iterations is a scale-space map which we propose to call *Iterative Gaussian scale-space map*. The map consists of a series of dot patterns some of which grow reaching the maximum iteration scale and some other terminate as the iteration scale increases. A pair of dot patterns may grow and merge to become a single dot pattern which may reach the maximum iteration scale. The dot patterns that survive the maximum number of iterations are indicative of those local maxima which are detected at all levels of detail, fine as well as coarse. The dot patterns that appear but terminate after a number of iterations are indicative of those local maxima that are detected at fine levels of detail but disappear as the degree of smoothing increases. The dot patterns that appear at all scales are the more compelling determiners of the global shape of a curve than those that appear at fine scales but disappear as the iteration scale increases. As the iterations proceed the

dot patterns either remain stationary or interact with each other. Two neighboring dot patterns may either attract or repel.

A number of digital curves and their scale-space map are shown in Figures 10.1 through 10.4. The starting point is indicated with an ↗ on each digital curve and the curve is described in the clockwise direction. The scale-space map of each curve consists of an aggregate of dot patterns some of which persist surviving the maximum iteration scale and some other disappear as the iteration proceeds. Two dot patterns may interact either attracting or repelling each other. Some of the dot patterns merge to become a single dot pattern which may persist and reach the maximum number of iteration.

10.6 Scale-space property

The digital Gaussian filter coefficients c_{-1}, c_0 and c_1 satisfy the scale-space property which demands that the number of local maxima of absolute curvature should not increase as the iteration scale increases. This conjecture follows from a proposition presented in [27] which states that a three-kernel with positive elements c_{-1}, c_0 and c_1 is a scale-space kernel if and only if $c_0^2 \geq 4c_1c_{-1}$. We find that the coefficients c_{-1}, c_0 and c_1 are all positive and the relation $c_0^2 \geq 4c_1c_{-1}$ is satisfied. As the digital curve is iteratively convolved with the same kernel, at each iteration the scale-space property is preserved.

10.7 Scale-space behavior of corner models

In this section we propose to make an analysis of the scale-space behavior of isolated corner models, namely Γ model, END model and STAIR model as presented in the last chapter. The models being open curves, the curvature measurements on them are affected by the ends of the curve. In order to avoid the end effects the arm length of the models (given by the number of points on the leading and the trailing side of the model) is taken to be sufficiently large (each side of the model should consist of at least 100 points) so that 100 iterations can be performed on each model. The search for extrema is restricted in the neighborhood of the angular points of the

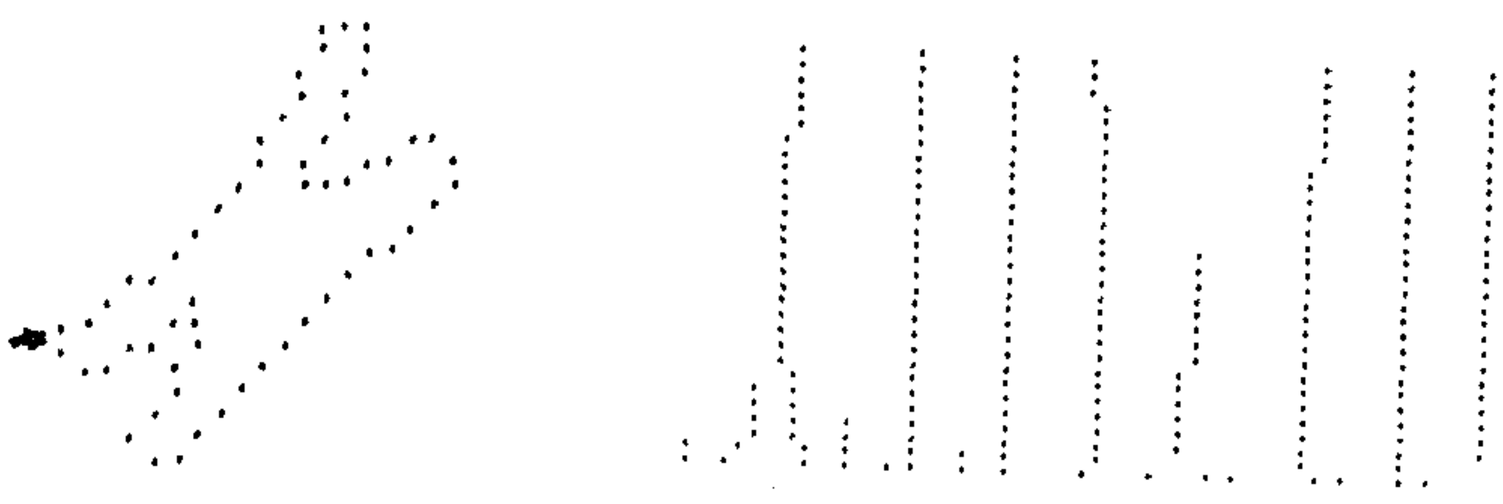


Figure 10.1: The chromosome shaped curve (top left), its scale-space map (top right) and tree organization (bottom).

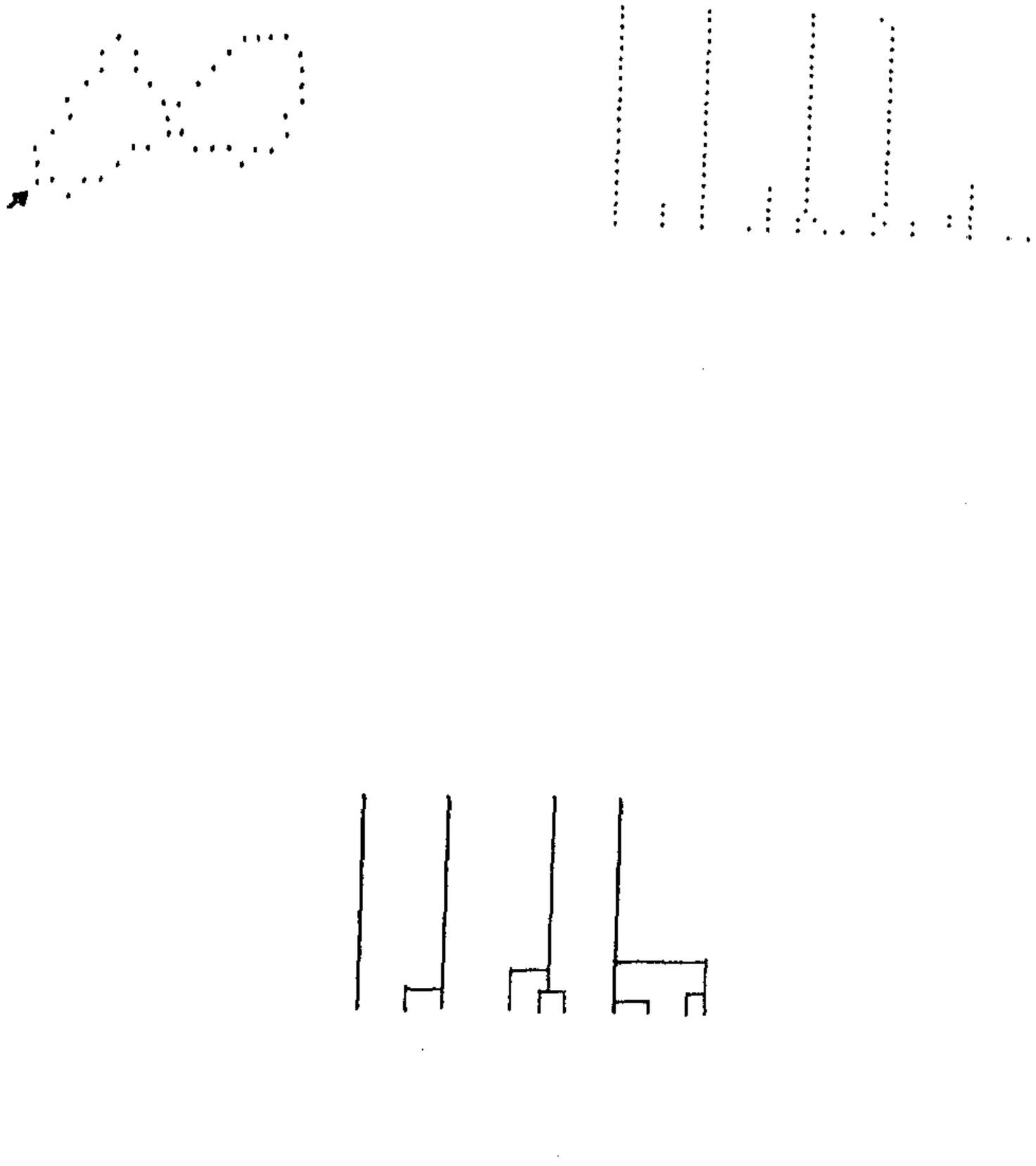


Figure 10.2: The figure-8 curve (top left), its scale-space map (top right) and tree organization (bottom).

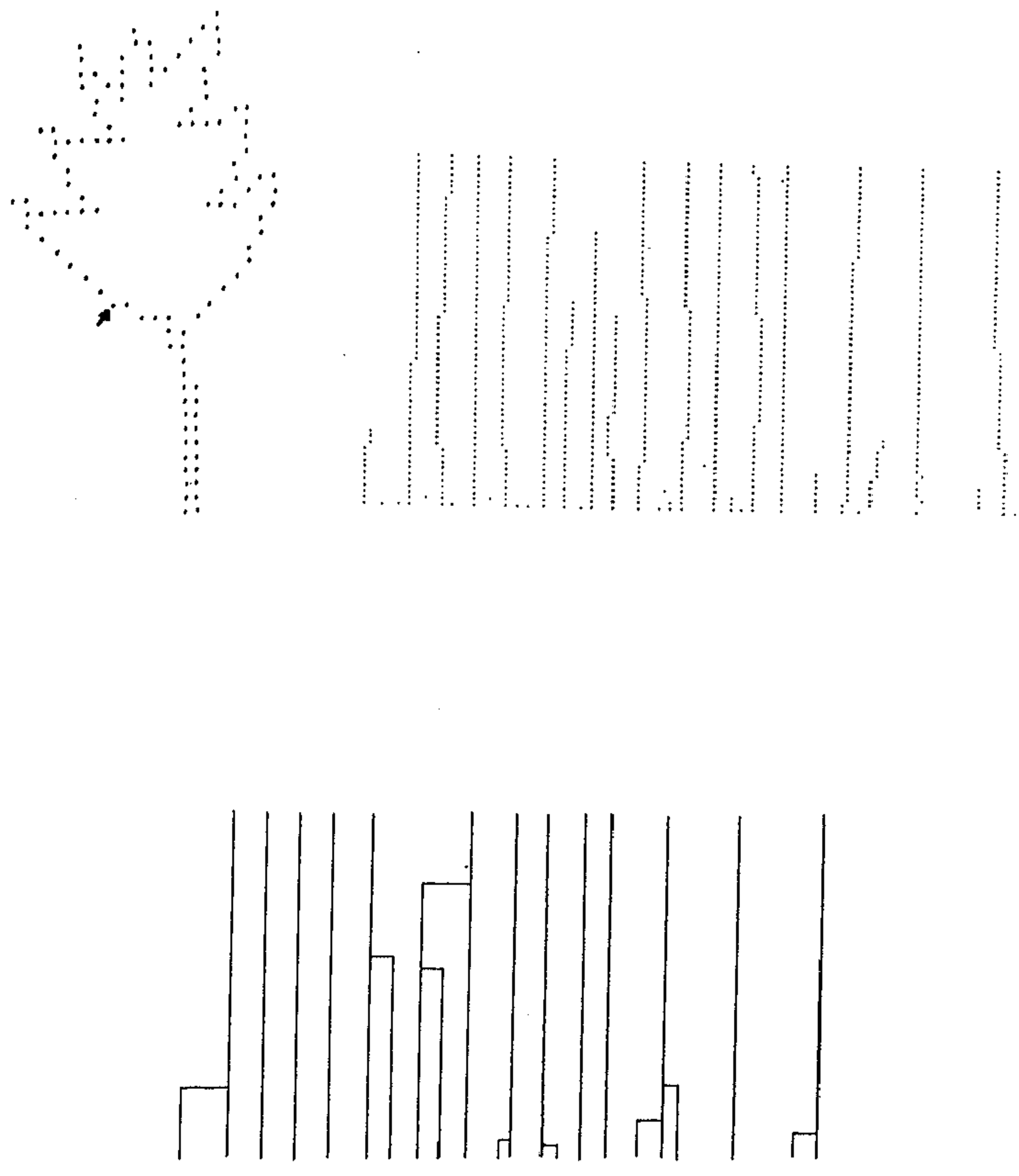


Figure 10.3: The leaf-shaped curve (top left), its scale-space map (top right) and tree organization (bottom).

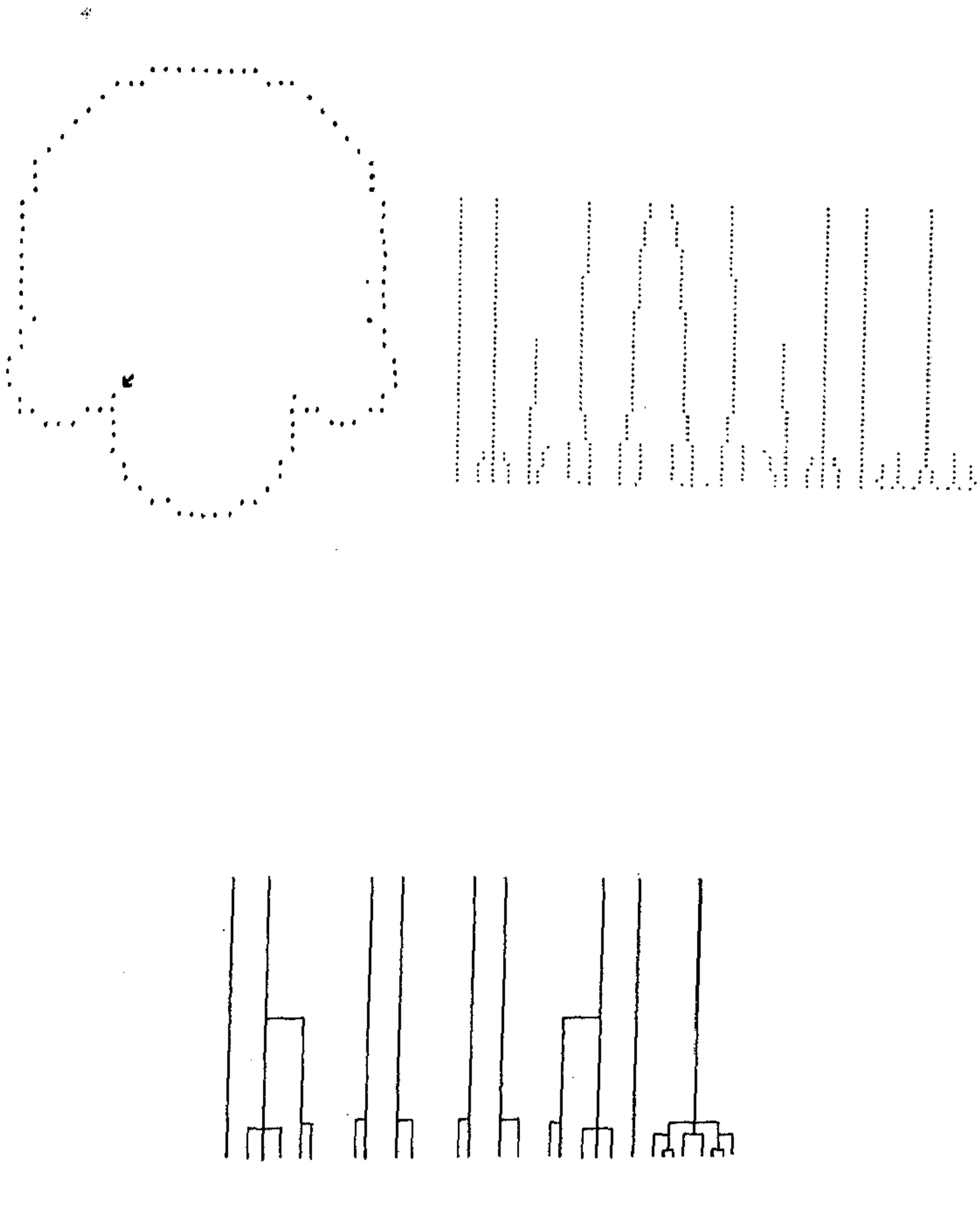


Figure 10.4: The curve with four semi-circles (*top left*), its scale-space map (top right) and tree organization (*bottom*).

model. In order to avoid the spurious numerical absolute curvature maxima, a very small input threshold 0.0001 is used. An absolute curvature maxima is regarded as a true maximum if the absolute curvature exceeds 0.0001. The scale-space map of a model is constructed neglecting the spurious numerical maxima of absolute curvature and restricting the search for true extrema near the angular point(s) of the models. The map shows the location of the true absolute curvature maxima over the iteration scales.

Figure 10.5 shows Γ models with different included angles and their scale-space map. As seen from these figures, the scale-space map of a Γ model with included angle $\pi/2$ consists of a single persistent and stationary dot pattern located at the angular point of the model. The scale-space map of a Γ model with included angle of either $\pi/4$ or $3\pi/4$, too consists of a single persistent dot pattern located at the angular point of the model. The dot pattern is not stationary. It exhibits movement as the iteration scale increases.

Figure 10.6 shows END models with different included angles and their scale-space map. The scale-space map of an END model with both the included angles equal to $\pi/2$ (Figure 10.6(a)) initially consists of two dot patterns each located at the angular points of the model. But as the iteration proceeds, the two dot patterns merge to become a single persistent dot pattern. The persistent dot pattern is stationary if the number of points forming the width of the END model is odd, otherwise it exhibits an oscillatory movement (see Figure 10.6(b)). Figure 10.6(c) shows another END model with both the included angles equal to $\pi/2$ but the width of this model is half the width of the model in Figure 10.6(a). The scale-space map of this model exhibits the same behavior as that of the former, but as the width is smaller the dot patterns of its scale-space map merge faster than those of the former. Figure 10.6(d) shows an END model with both the included angles equal to $\pi/4$ and 10.6(e) shows another END model with both the included angles equal to $3\pi/4$. The scale-space map of either of these models is similar to that of the END model in Figure 10.6(a). Figure 10.6(f) shows an END model with one angle equals to $\pi/4$ and the other $\pi/2$. The scale-space map initially shows two dot patterns located at the angular points of the model. As the iteration proceeds the dot patterns attract each other but they do not merge. The dot pattern at the

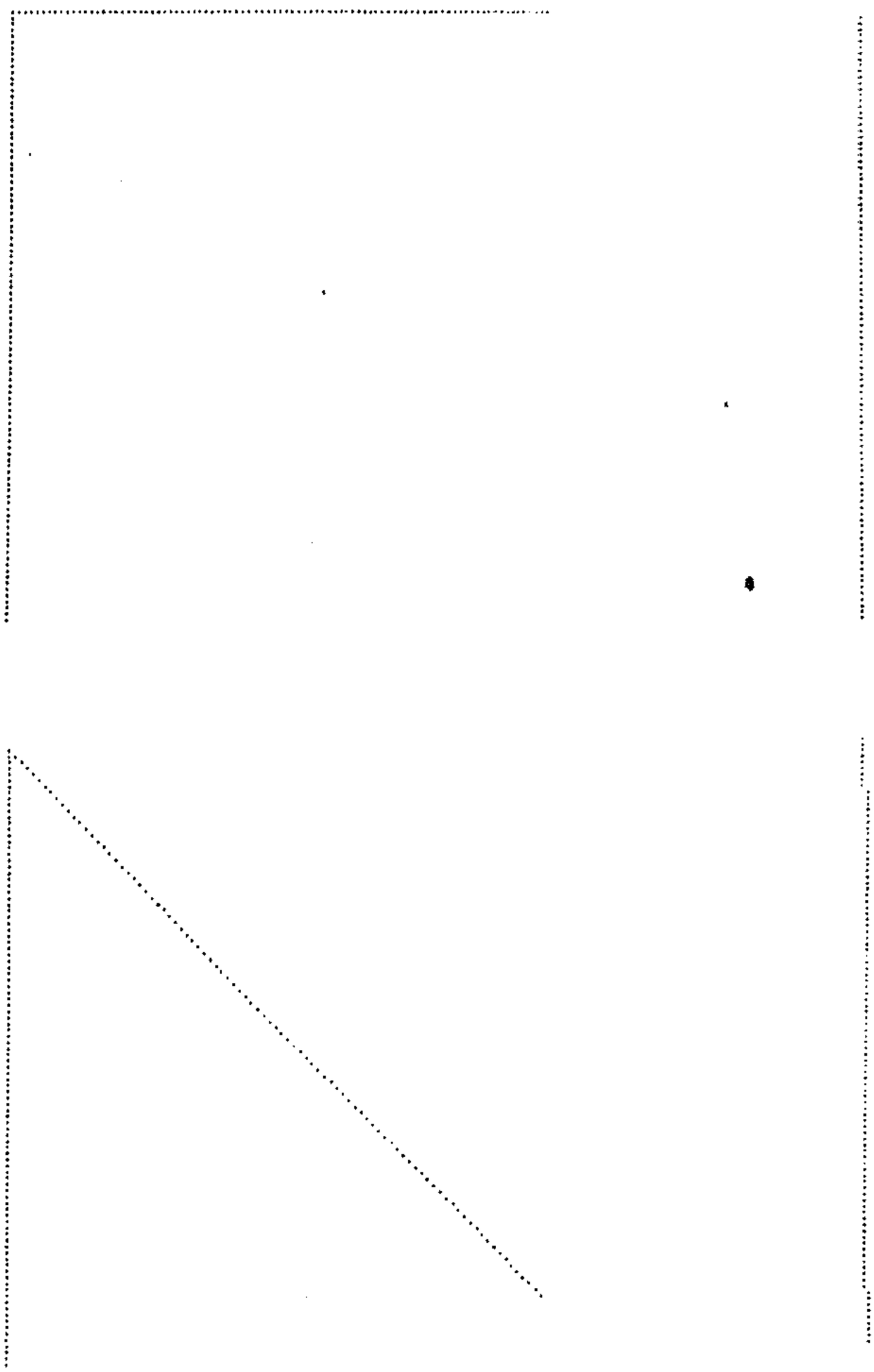


Figure 10.5: Γ models and their scale-space map. The left figure is the model and the right is its scale-space map.

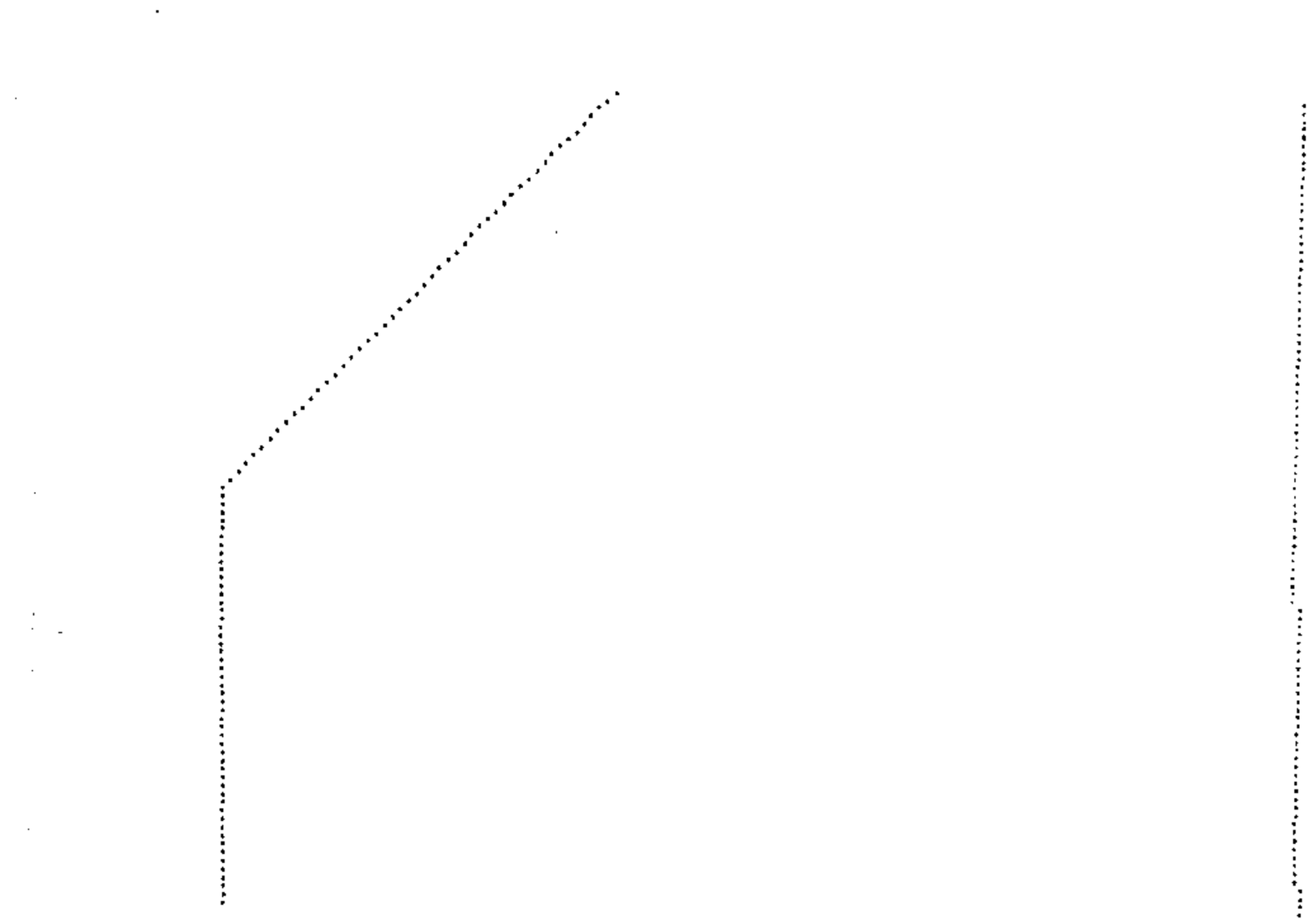
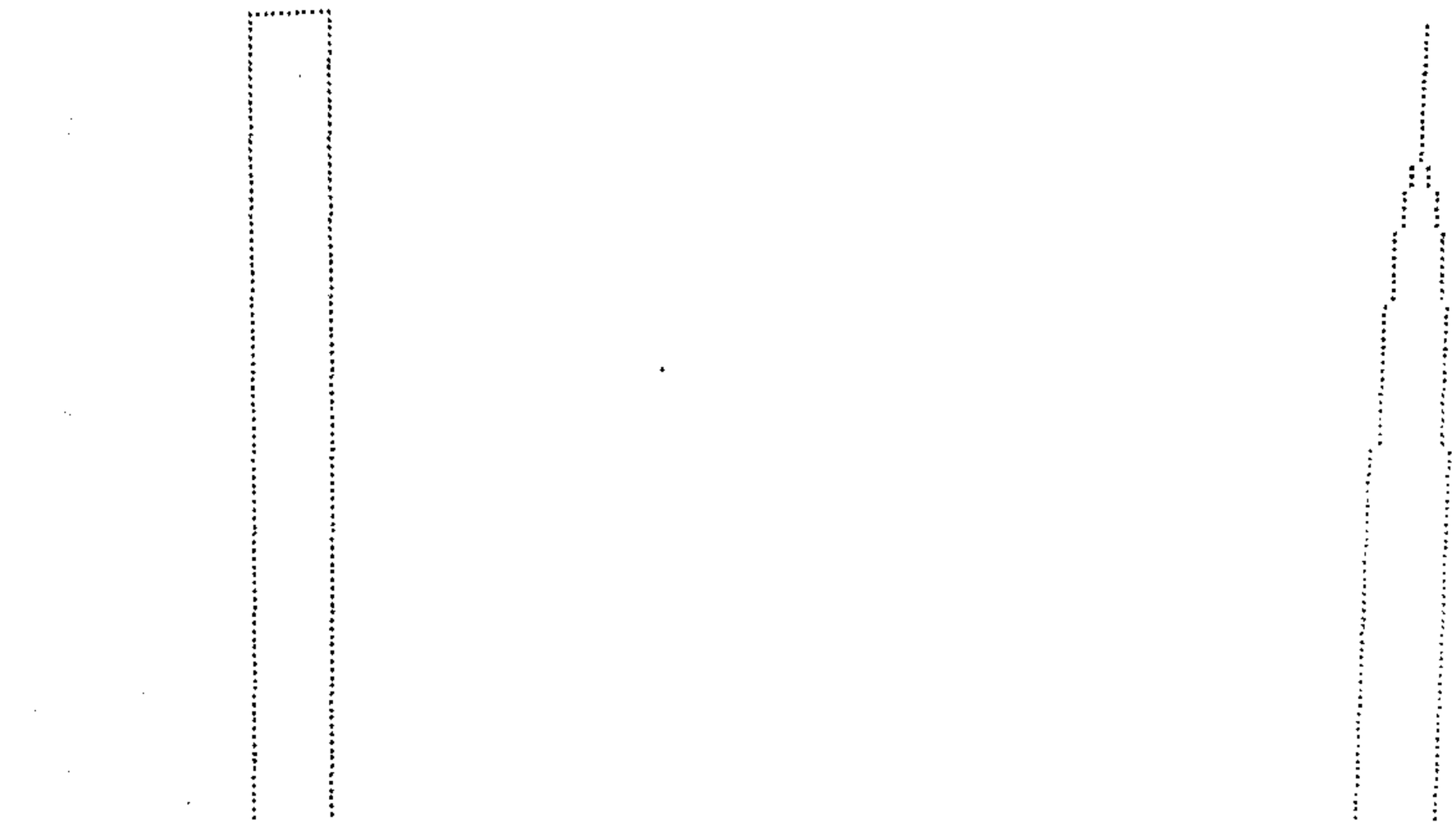


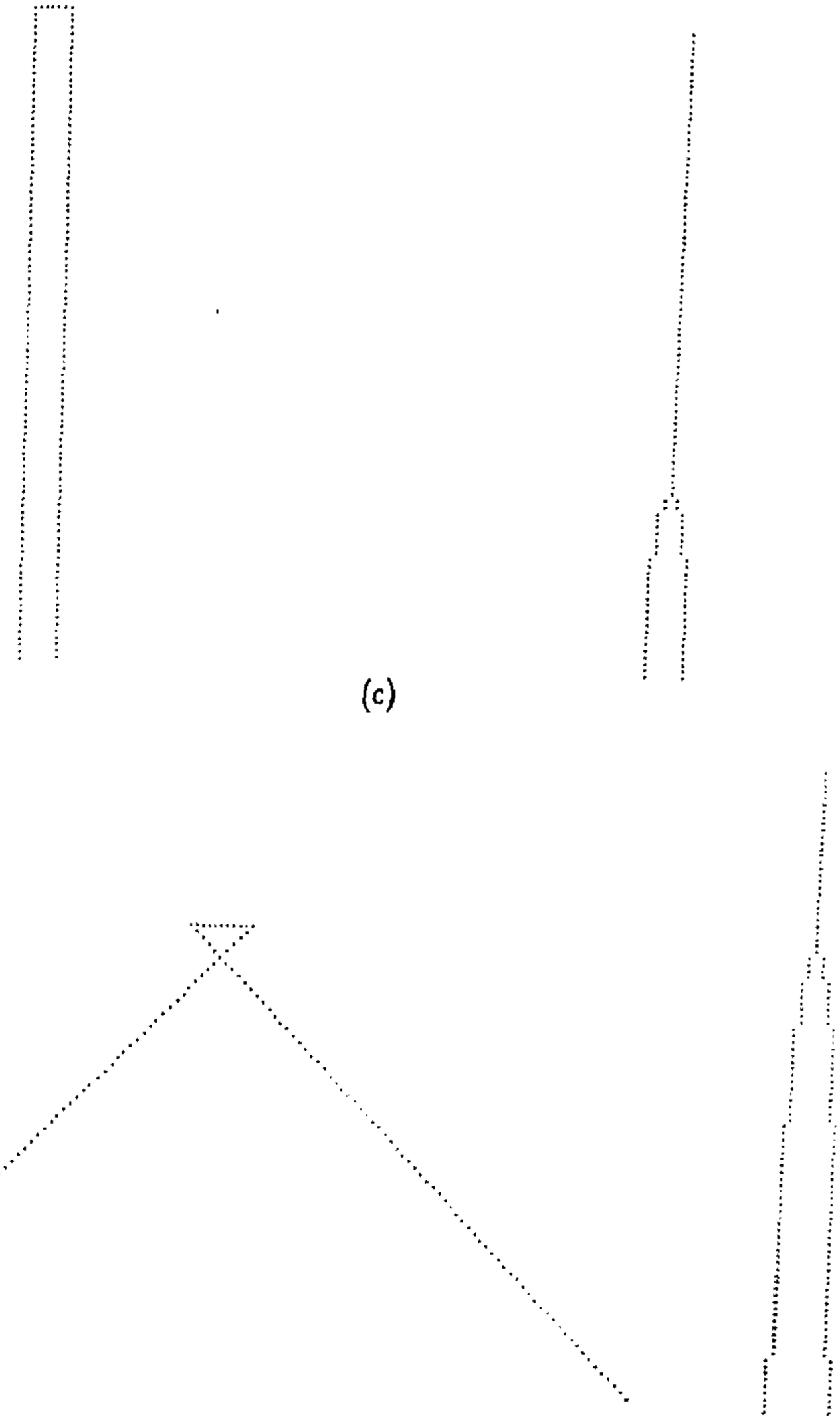
Figure 10.5: Continued.



(a)

(b)

Figure 10.6: END models and their scale-space map. The left figure is the model and the right is its scale-space map.



(c)

(d)

Figure 10.6: Continued.

Figure 10.6: Continued.

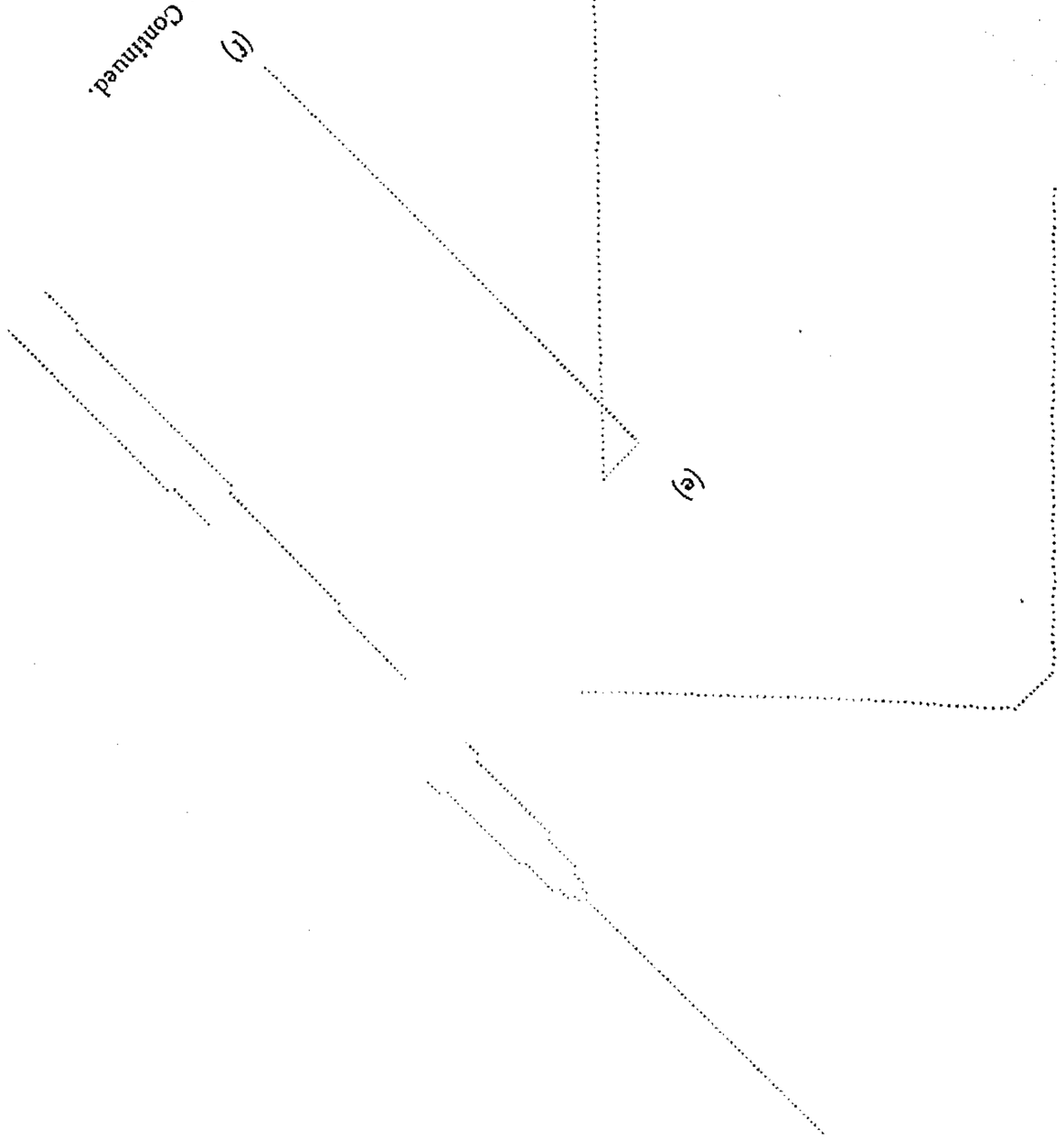
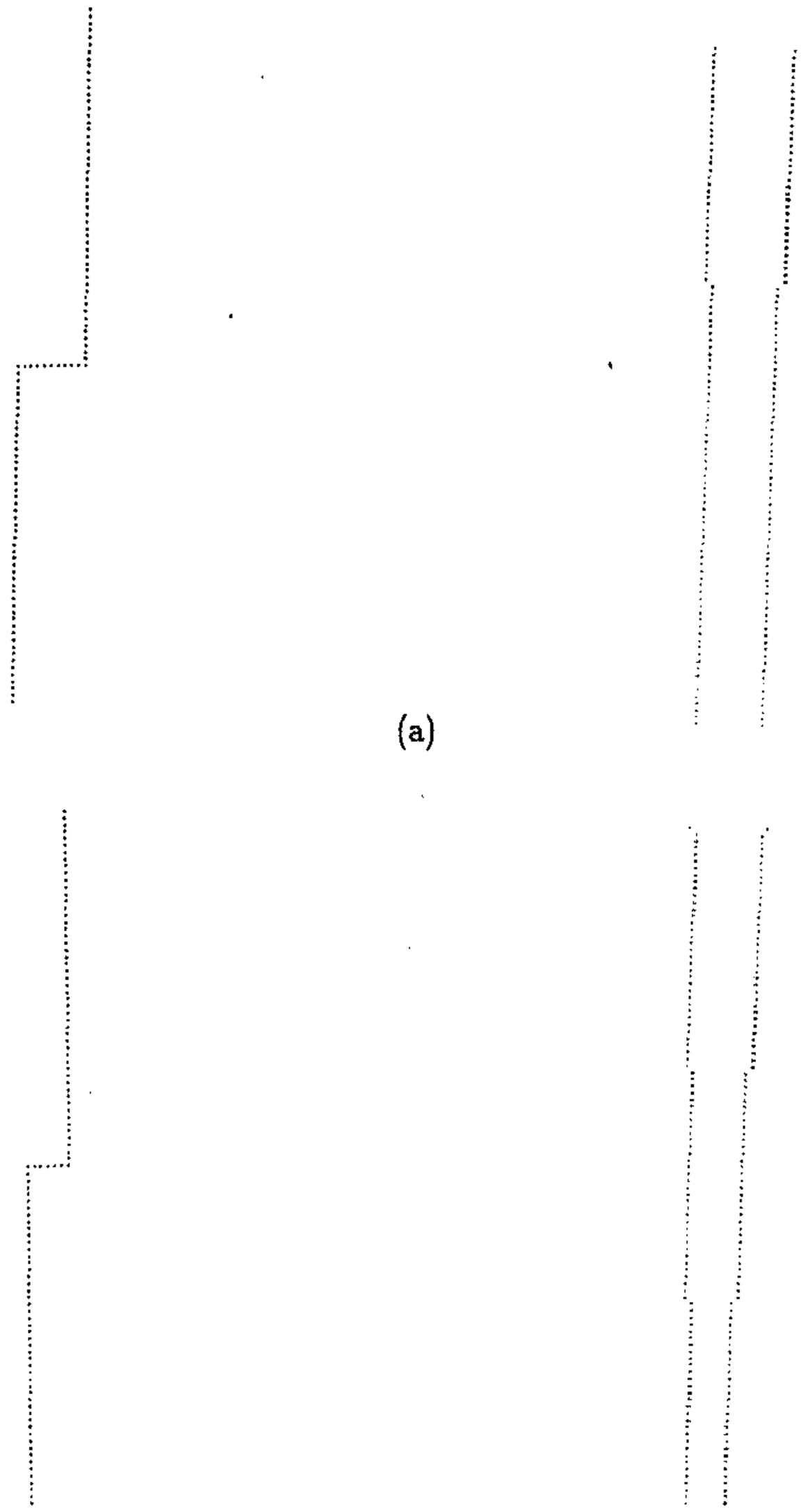


Figure 10.6. Continued.

(h)

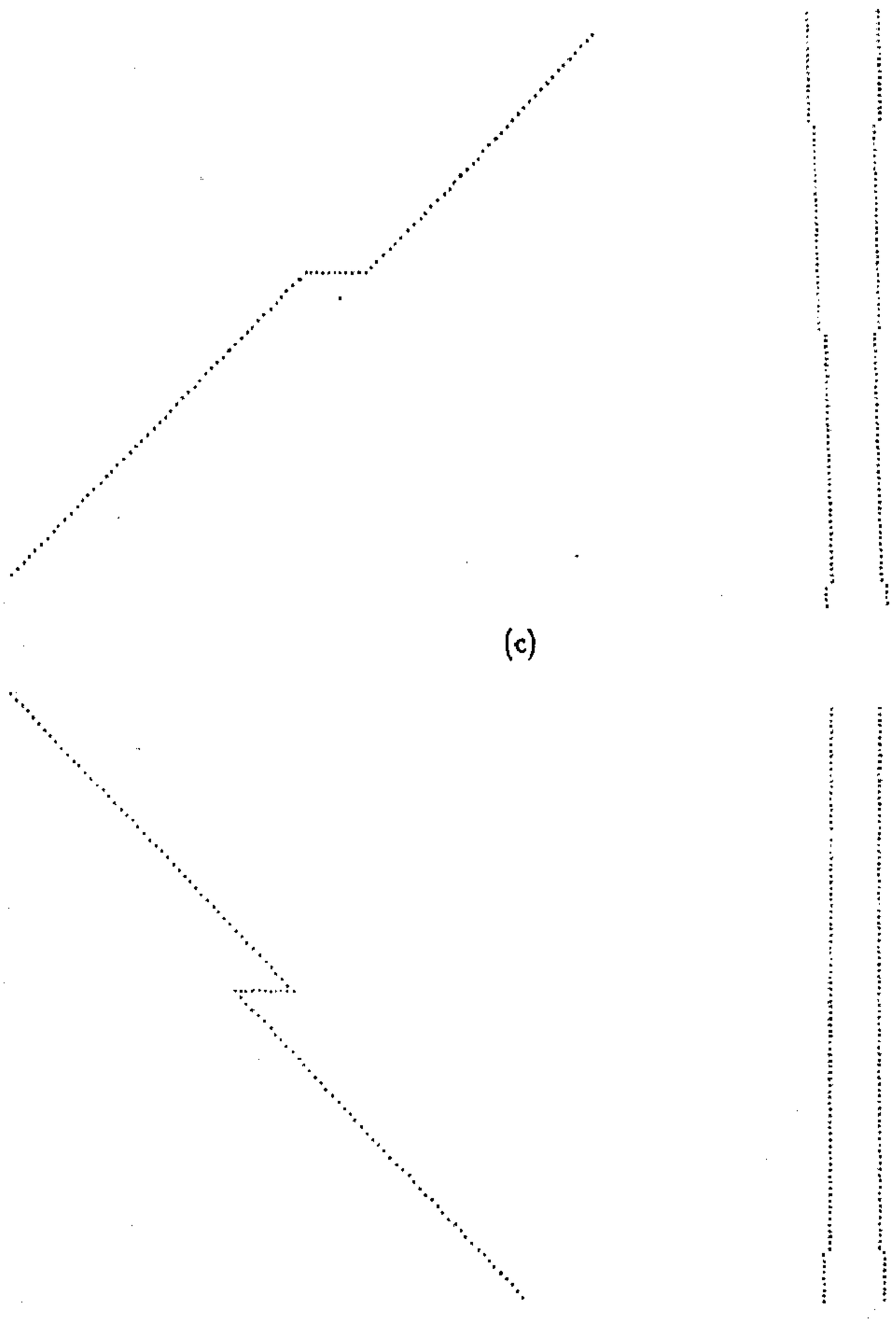
(i)



(a)

(b)

Figure 10.7: STAIR models and their scale-space map. The left figure is the model and the right is its scale-space map.



(c)

(d)

Figure 10.7: Continued.

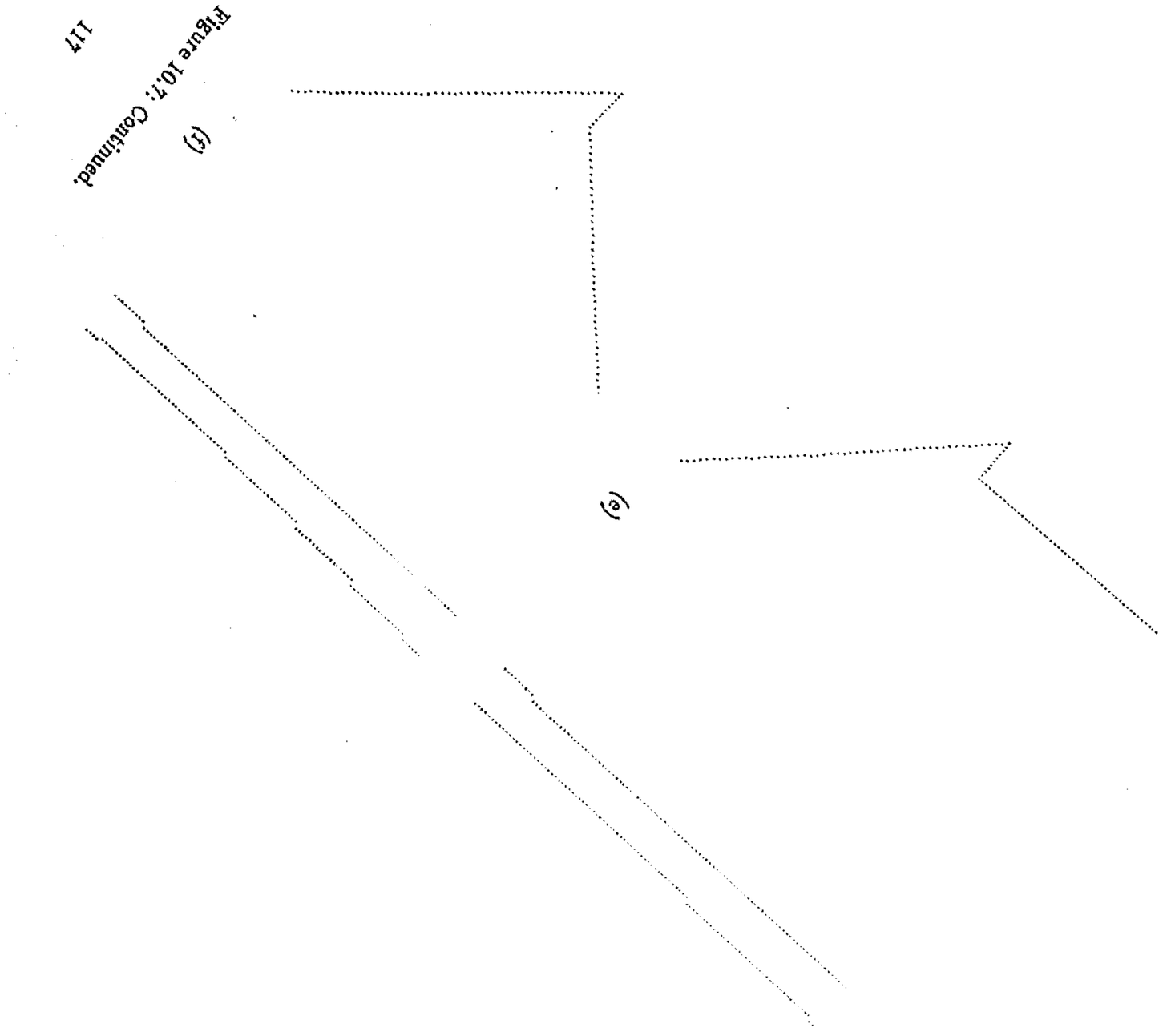
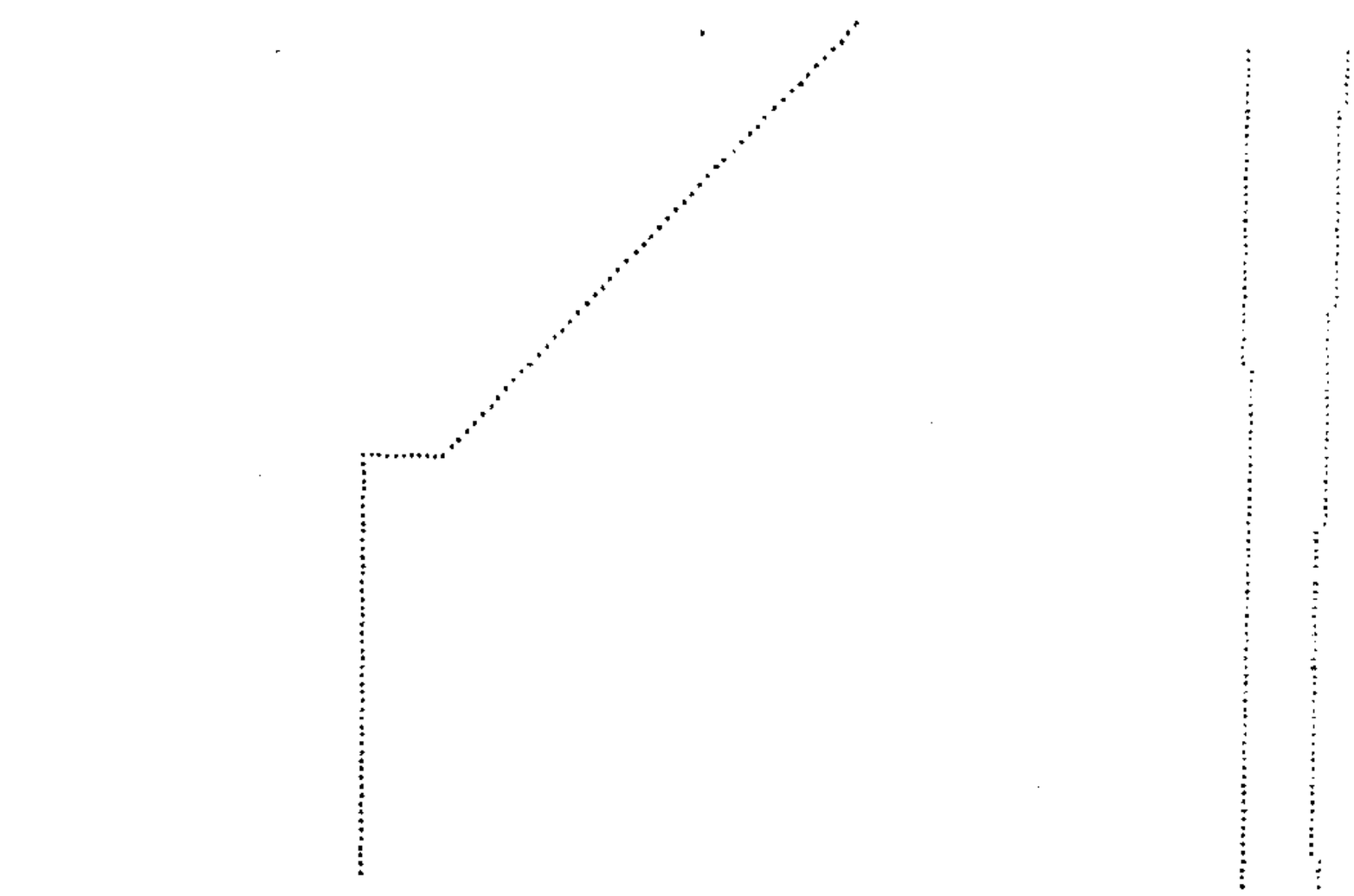


Figure 10.7: Continued.



(g)

Figure 10.7: Continued.

weaker corner (angle= $\pi/2$) terminates whereas the other persists. Figure 10.6(g) shows an END model with one angle $\pi/4$ and the other $3\pi/4$. Figure 10.6(h) shows another END model with one angle $\pi/2$ and the other $3\pi/4$. The scale-space map of either of these models is similar to that of the model in Figure 10.6(f).

Figure 10.7 shows STAIR models with different included angles and their scale-space map. Figure 10.7(a) shows a STAIR model with both the included angles equal to $\pi/2$. The scale-space map consists of two persistent dot patterns located at the angular points of the model. The dot patterns repel each other as the iteration scale increases. Figure 10.7(b) shows another STAIR model with both angles equal to $\pi/2$ but with width half that of the model in Figure 10.7(a). The scale-space map of the model is almost similar to that of the model in Figure 10.7(a) but as the width of this model is smaller the dot patterns repel earlier. Figure 10.7(c) shows a STAIR model with both the included angles equal to $3\pi/4$. The scale-space map consists of two persistent dot patterns each located at the angular points of the model. The dot patterns initially attract each other but subsequently repel as the iteration proceeds. Figure 10.7(d) shows another STAIR model with both angles equal to $\pi/4$. The scale-space map consists of two persistent dot patterns located at the angular points of the model. The dot patterns initially attract each other but remain separated without being merged as the iteration proceeds. Figure 10.7(e) shows a STAIR model with one angle equals to $\pi/4$ and the other $\pi/2$, Figure 10.7(f) shows a STAIR model with one angle equals to $\pi/4$ and the other $3\pi/4$ and Figure 10.7(g) shows a STAIR model with one angle equals to $\pi/2$ and the other $3\pi/4$. The scale-space map of each of these models consists of two persistent dot patterns located at the angular points of the model. The dot patterns initially attract eacn other for once only but subsequently repel as the iteration proceeds.

From the above discussion we conclude that the scale-space map of a Γ model consists of a single persistent dot pattern which may either be stationary or may exhibit movement as the iteration scale increases. The scale-space map of an END model initially consists of two dot patterns located at the angular points of the model. If both the included angles be equal then as the iteration proceeds, the two dot patterns attract each other and merge to become a single dot pattern which may be stationary. If the angles be different then as the iteration proceeds

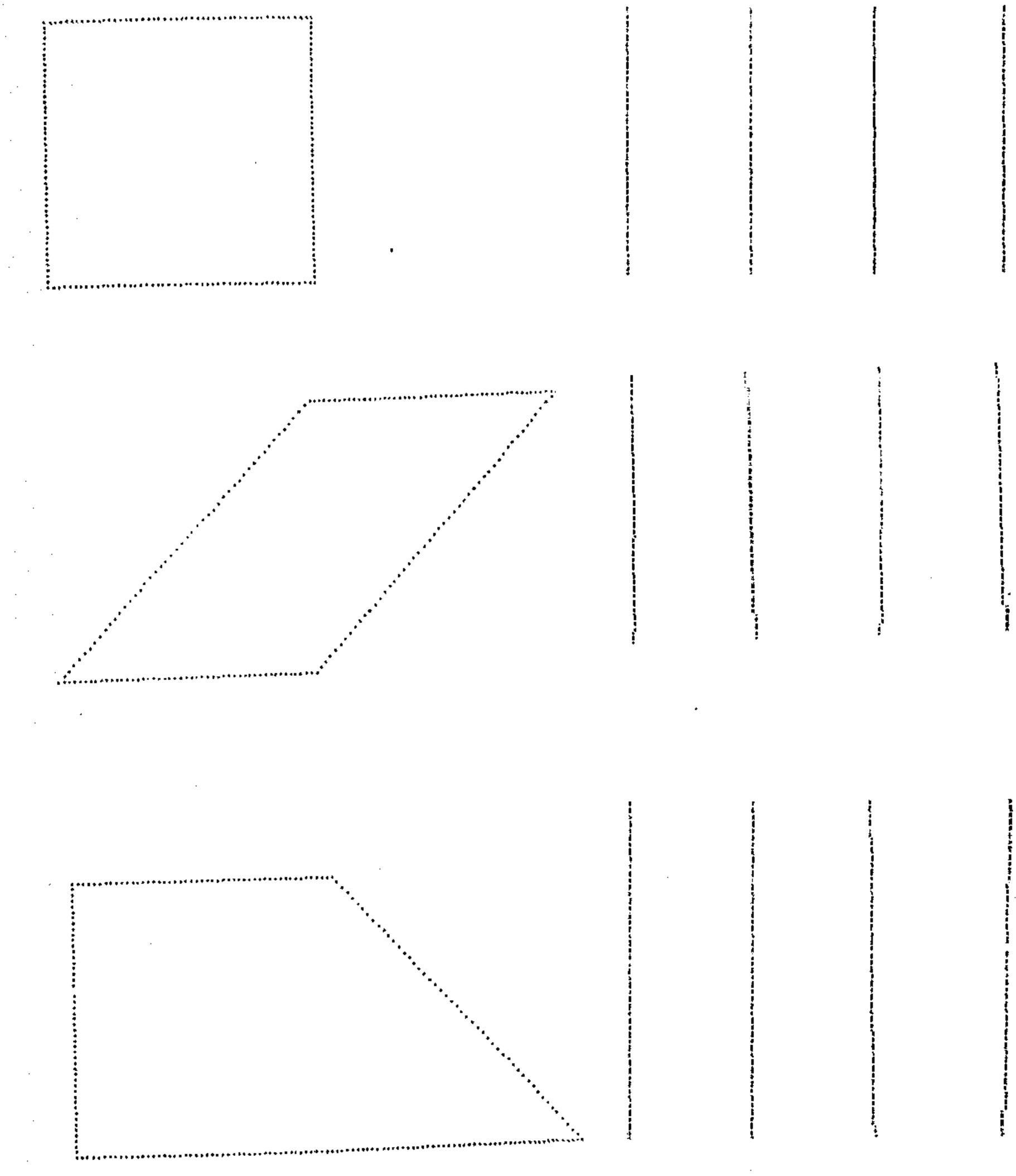


Figure 10.8: A number of polygons and their scale-space map.

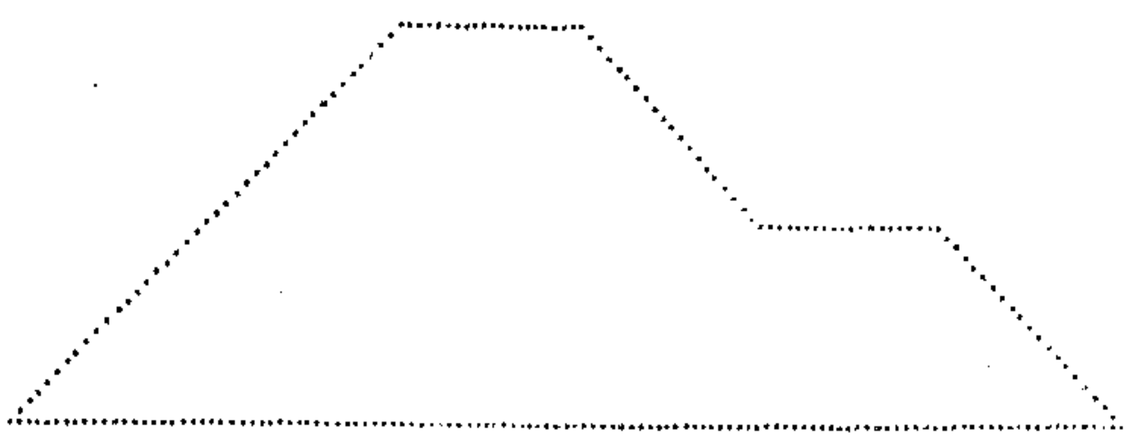


Figure 10.8: Continued.

though the dot patterns initially attract each other but the one at the weaker corner terminates and the other persists. The scale-space map of a STAIR model consists of two persistent dot patterns located at the angular points of the model. The dot patterns may either repel each other or may initially attract each other for once only but subsequently repel. The dot patterns do not merge nor do they disappear.

The models that have been analysed are single and double corner models. There is no explicit analysis of models that consist of more than two corners. In order to give the reader a perspective of the scale-space behavior of models that consist of more than two corners, a number of polygons and their scale-space map are shown in Figure 10.8. These maps are in conformity with the scale-space behavior of the models that have been analysed.

10.8 Tree organization and corner detection

To detect corners on digital curves, a tree representation similar to that in [61] and [39] is constructed from the scale-space map of a curve. The dot patterns that are caused by extremely low absolute curvature (below 0.0001) are removed from the map. After this cleaning process, for each dot pattern a vertical line located at the finest scale location of the dot pattern is drawn. If two dot patterns merge to become a single dot pattern then two vertical lines located at the finest scale location of the two dot patterns are drawn and a single vertical line located at the merging scale of the two dot patterns is also drawn. The three vertical lines are joined by a horizontal line. Each of the non-surviving vertical lines is joined by a horizontal line to its nearest immediately more persistent vertical line. The persistency of a vertical line is measured by its height from the $x-$axis. A non-surviving vertical line L_1 (say) is joined to its nearest vertical line L_2 (say) if and only if the height of L_2 from the $x-$axis is immediately greater than that of L_1. This tree construction method is based on the assumption that each dot pattern is influenced most by its nearest and immediately more persistent dot pattern. The tree organization of two END models are shown in Figure 10.9. The tree organization of a number of digital curves are shown in Figures 10.1 through 10.4. We note that this method of construction of tree is simpler than that in [39].

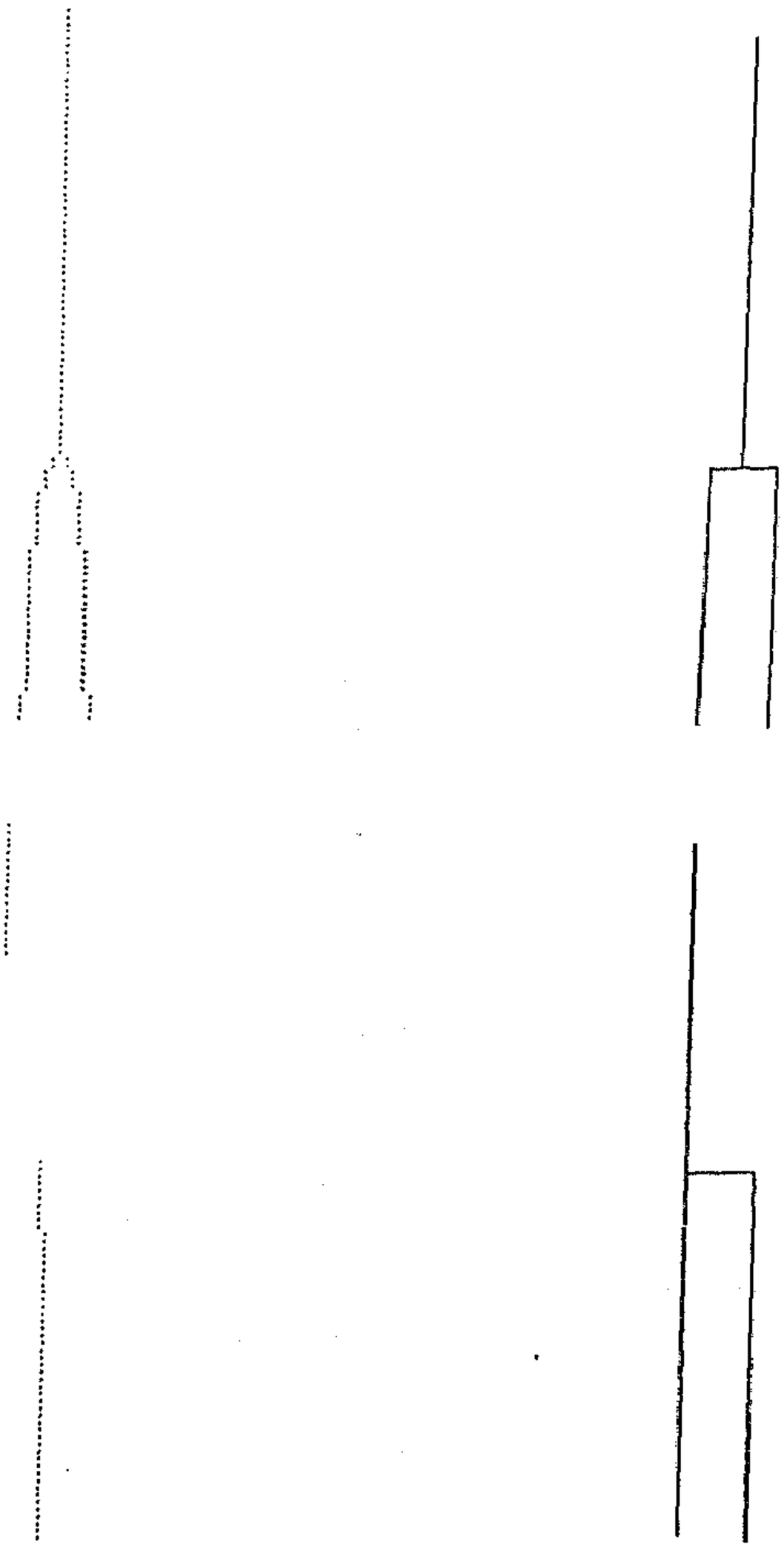


Figure 10.9: Tree organization of the scale-space map of two END models. The left figure is a scale-space map and the right its tree organization.

A tree consists of a root and numerous branches and leaves. To parse a tree the length of its root, branches and leaves are measured. A tree consisting of a single root only corresponds to a corner point located at the foot of the root. For a tree with offspring the length of the root is compared with the height of its offspring which is defined as the vertical distance between the branching point of the tree and the $x-$axis. If the root length does not fall below the height of the offspring then the root only corresponds to a corner located at the point of orthogonal projection of the root. Otherwise the search for corners proceeds to the offspring. If an offspring is a leaf whose length is greater than that of its parent then the offspring corresponds to a corner located at the location of the leaf at the finest scale. If an offspring has its own offspring then the search for corner is applied to this family of tree and the searching process is similar to that already described for a tree with offspring. This searching process continues until all branches and leaves of a tree are evaluated. Each tree of the tree organization of a scale-space map is evaluated in the similar manner. This tree parsing method has similarities as well as dissimilarities to that in [39]. In Figures 10.1 through 10.4 the limbs of the tree organization representing corners are indicated by bold lines.

10.9 Space requirements and computational load

In the Rattarangsi-Chin algorithm [39] as the varying window size is used the space requirements by the Gaussian filter coefficients is different for different window size. The space requirements (s) is exactly equal to the window size (w). The window size is varied in arithmetic progression whose first term is 3, common difference is 2 and the last term is is $n-1$, when n is even and n, when n is odd. The number of terms of the progression is $n/2-1$, when n is even and $(n-1)/2$, when n is odd. So the total space requirements by the filter coefficients is

$$\begin{aligned} \sum s &= (n-2)(n+2)/4, \quad \textit{when } n \textit{ is even} \\ &= (n-1)(n+3)/4, \quad \textit{when } n \textit{ is odd.} \end{aligned} \tag{10.21}$$

In the present approach as the constant window size $w = 3$ is used, the space requirements by the filter coefficients is small and finite ($= 3$). This shows that the

present approach reduces the space requirements considerably. In the Rattarangsi-Chin algorithm the space requirements is $O(n^2)$ whereas, in the present approach the space requirements does not depend on the data size, it is small and finite.

To determine the computational load we note that the smoothing process in [39] and in the present approach are parallel in nature. So in order to compare the computational load of either of the smoothing processes, it is sufficient to determine the same at each point. In the Rattarangsi-Chin smoothing process, for a window size of $w = 3$, the number of multiplications(m) and additions(a) required are $m = 3$ and $a = 2$, for a window size of $w = 5$, $m = 5$ and $a = 4$, for a window size of $w = 7$, $m = 7$ and $a = 6$ and so on for higher window size. For a window size of $w = 2j + 1$ which should not exceed the length of the curve, the number of multiplications required is $m = 2j + 1$ and the number of additions is $a = 2j$. This shows that the number multiplications as well as additions each form an arithmetic progression. For a digital curve with n points the number of terms in each series is $(n-1)/2$, when n is odd and $n/2 - 1$, when n is even. So the total number of multiplications and additions required are

$$\begin{aligned} \sum m &= (n-1)(n+3)/4, \quad \textit{when } n \textit{ is odd} \\ &= (n-2)(n+2)/4, \quad \textit{when } n \textit{ is even} \end{aligned} \tag{10.22}$$

and

$$\begin{aligned} \sum a &= (n-1)(n+1)/4, \quad \textit{when } n \textit{ is odd} \\ &= n(n-2)/4, \qquad \textit{when } n \textit{ is even.} \end{aligned} \tag{10.23}$$

So the total number of arithmetic operations required at the smoothing stage is $(n^2 + n - 2)/2$, when n is odd and it is $(n^2 - n - 2)/2$, when n is even. So the computational load of the smoothing process in [39] is $O(n^2)$ at each point.

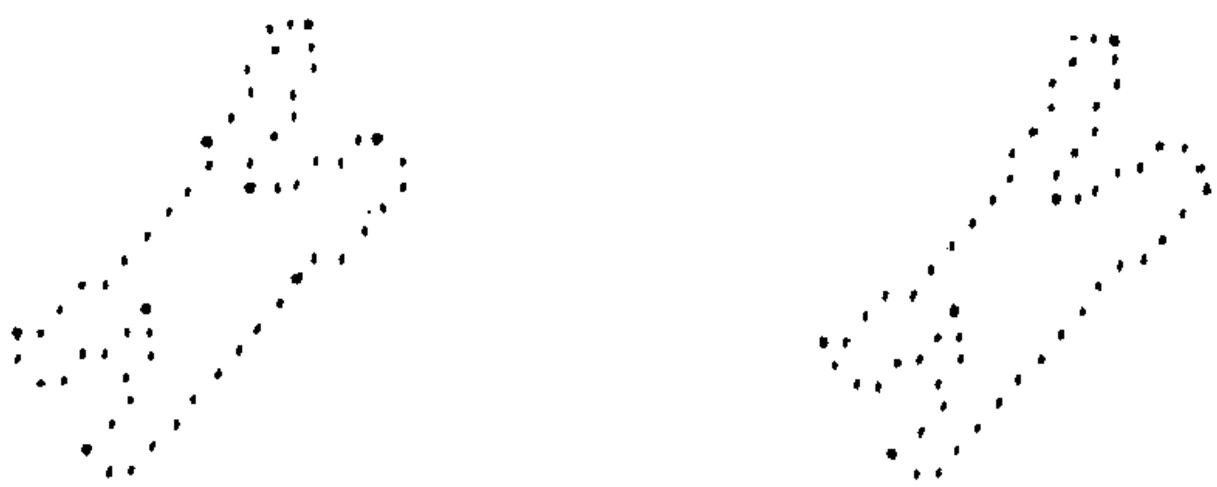
In the present approach at each iteration and at each point, the number of multiplications and additions required are $m = 3$ and $a = 2$. It has already been shown that the number of iterations to be performed on a digital curve is $(n-1)/2$, when n is odd and it is $n/2 - 1$, when n is even. So the total number of arithmetic operations required by the iterative smoothing process is $(5n-5)/2$, when n is odd

and it is $(5n - 10)/2$, when n is even. So the computational load of the iterative smoothing process is $O(n)$ at each point.

10.10 Experimental results

The corner detector developed in this chapter is applied on the same digital curves as in the last chapter. The corner points are indicated by bold solid circles on each curve. These are shown in Figures 10.10 through 10.13. These figures also show the corner points as obtained by the Rattarangsi-Chin algorithm [39]. From these results it is found that the corner detector developed in this chapter detects more corner points than the Rattarangsi-Chin algorithm.

To show the robustness of the corner detector to noise, white Gaussian noise is added to the *digital leaf* making the noise level σ vary from 0.5 to 2.5 in an interval of 0.5. The noisy curves are shown in Figure 10.14 indicating the corner points on them by bold solid circles. From these results we find that as the noise level increases no additional corner point is detected. The location of the corner points on the noisy curves is comparable to that on the original image.

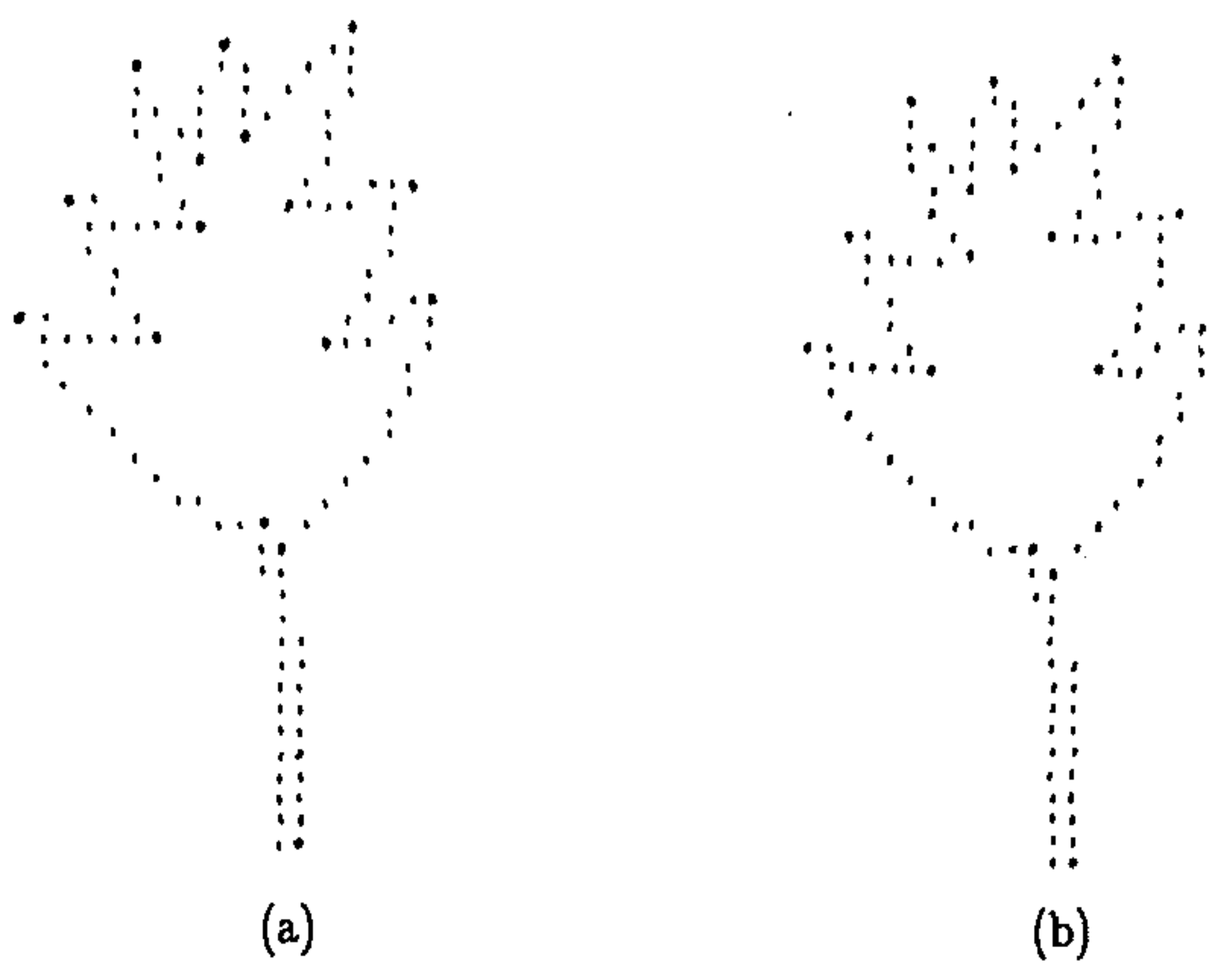


(a) (b)

Figure 10.10: The chromosome shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

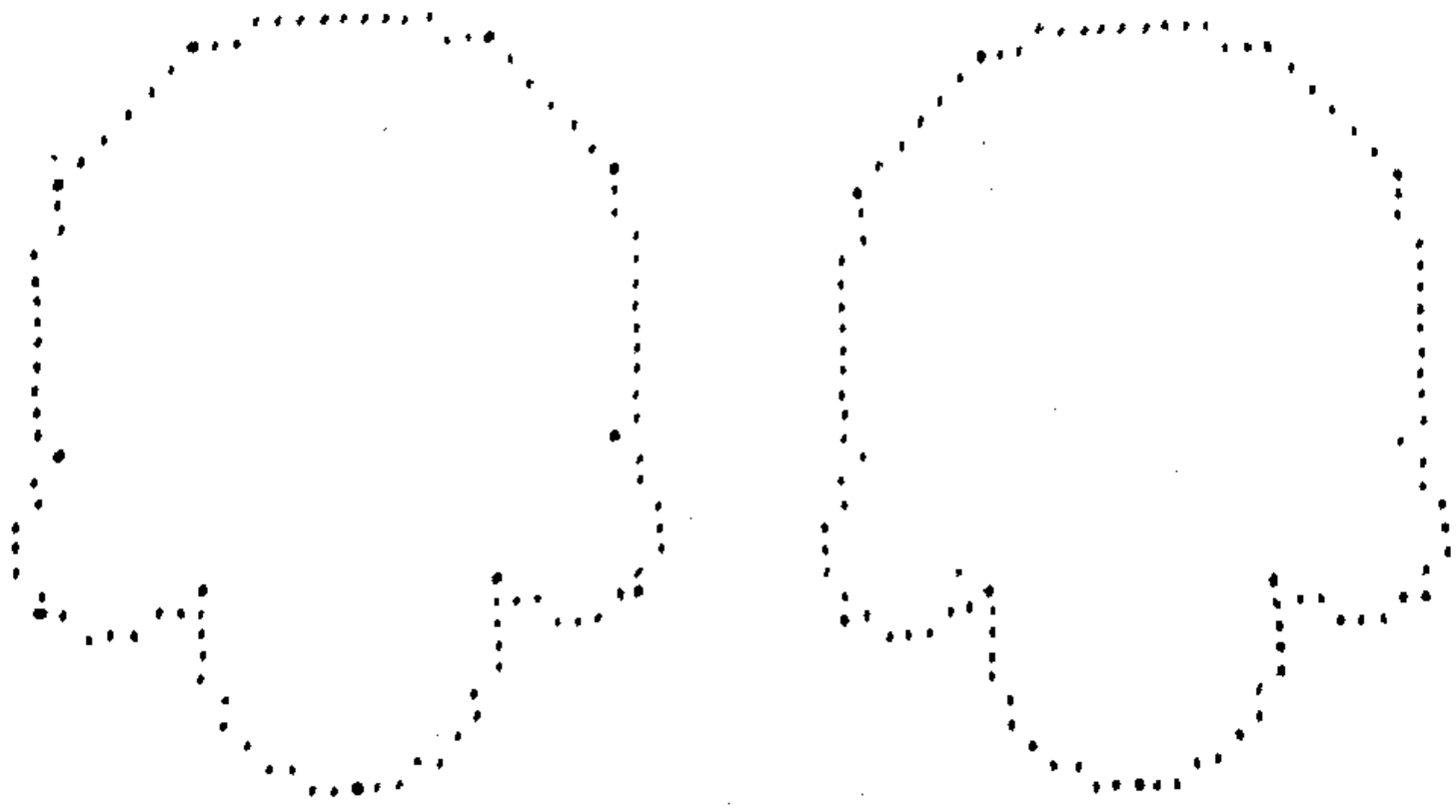
(a) (b)

Figure 10.11: The figure-8 curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.



(a) (b)

Figure 10.12: The leaf-shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.



(a) (b)

Figure 10.13: The curve with four semi-circles. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

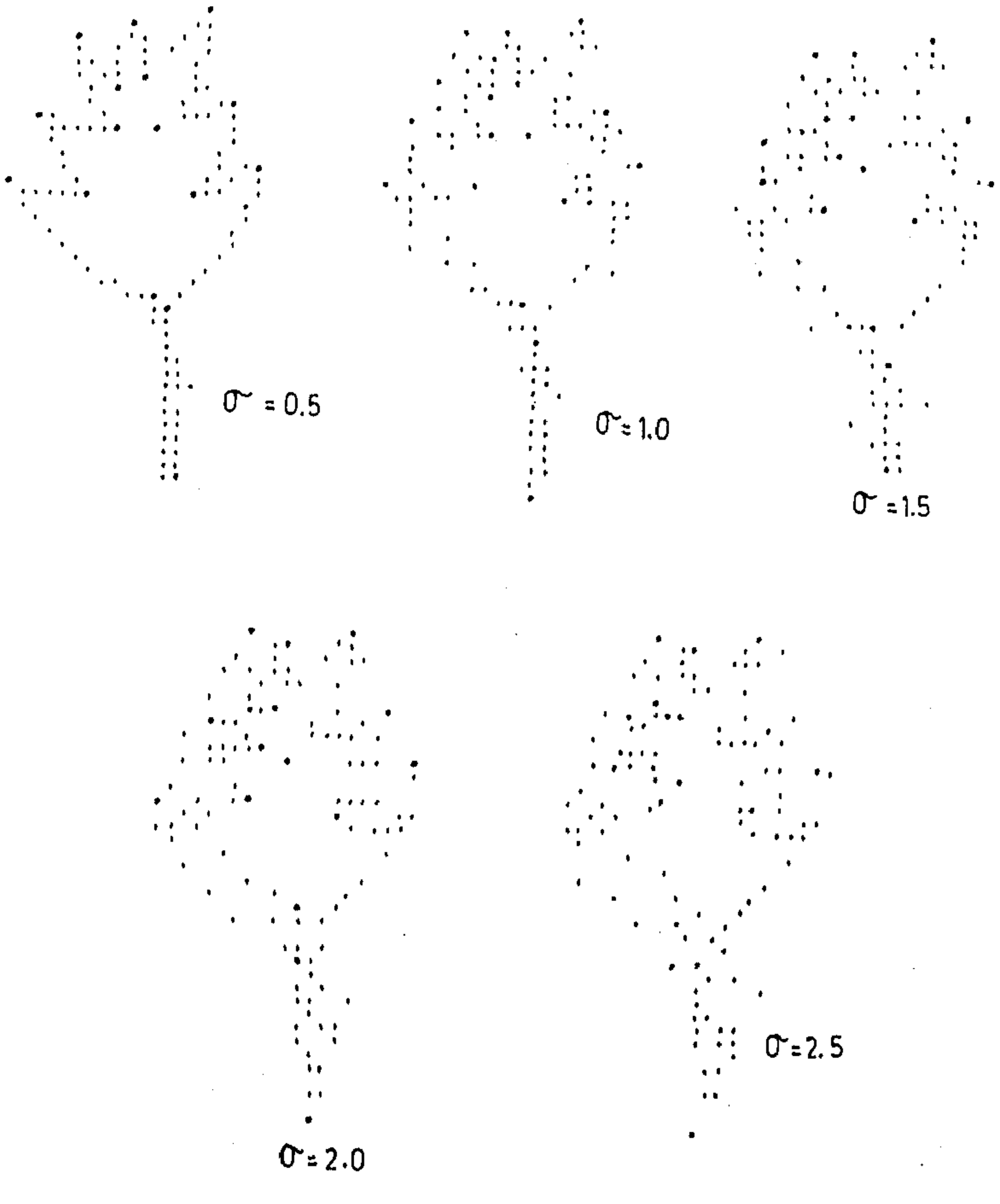


Figure 10.14: The noisy leaf at varying noise levels. Corners are indicated by bold solid circles.

Chapter 11

Scale-space analysis and corner detection using discrete scale-space kernel

The Gaussian kernel is by its very nature applicable to continuous signals and curves. Babaud et al. [7] has proved that in a broad class of functions Gaussian kernel enjoys scale-space property while convolving it with continuous signals and curves. Lindeberg [27] observes that in order to apply this kernel to digital signals / curves it is necessary to discretise it properly otherwise scale-space property may not be preserved. Rattarangsi and Chin [39] use the digital Gaussian filter coefficients of [10] namely, $c_{-1} = c_1 = 0.2236$ and $c_0 = 0.5477$ for window size $w = 3$. These filter coefficients have been mentioned in [29] and [11] as the best approximation of the Gaussian distribution for window size $w = 3$. The digital Gaussian filter coefficients for window size $w = 2j + 1,\ j > 1$ are generated by convolution of these coefficients with themselves for j times. Using discrete scale-space theory developed by Lindeberg [27] we have shown in the last chapter that the digital Gaussian filter coefficients used by Rattarangsi and Chin [39] do satisfy scale-space property. Since convolution of a scale-space kernel with another scale-space kernel generates a scale-space kernel [27] so the digital Gaussian filter coefficients for window size higher than $w = 3$ generated by Rattarangsi and Chin [39] do satisfy scale-space property.

The essence of scale-space theory is not only to preserve scale-space property but to keep a provision for allowing arbitrary amount of smoothing. In scale-space analysis with digital Gaussian filter coefficients with varying window size, each window length corresponds to a specific value of σ. If for two successive window length of w and $w+3$ the corresponding values of σ be σ_w and σ_{w+3} then it is not possible to impart a digital curve a degree of smoothing corresponding to an arbitrary σ satisfying $\sigma_w < \sigma < \sigma_{w+3}$. In the iterative Gaussian smoothing which we have presented in the last chapter each iteration corresponds to a specific degree of smoothing. Since the iteration scale is discrete in nature hence the amount of smoothing is also discretised. So in Gaussian smoothing with digital filter coefficients it is not possible to smooth a digital curve by an arbitrary amount, if desired.

Instead of discretising the Gaussian kernel Lindeberg [27] has developed a genuinely discrete theory for scale-space analysis of discrete signals and curves with a *continuous* scale parameter. Starting from a two-kernel having scale-space property he has obtained a generalized binomial kernel with finite support. He has also suggested the form of a discrete scale-space kernel with infinite support. Introducing the semi-group property and the symmetric property he has concluded that for discrete signals the most reasonable discrete scale-space kernel with a *continuous* scale parameter t is

$$T(n;t) = e^{-t} I_n(t), \quad t > 0 \tag{11.1}$$

and $I_n(t)$ is the modified Bessel function of integer order. In this chapter we propose to show the application of the kernel $T(n;t)$ to scale-space analysis of $2-D$ digital curves followed by its application to corner detection. A $2-D$ digital curve is decomposed into two $1-D$ signals. Each signal is convolved with the kernel $T(n;t)$. The extreme curvature points are located at varying t. The scale-space map showing the movement of the curvature extrema over varying t is used to detect and locate the corners. The numerical problems that arise in the implementation are addressed in sequence and solutions are proposed.

11.1 Generation of filter coefficients

To convolve a digital curve with the kernel (11.1) it is necessary to generate the filter coefficients $T(n;t) = T(-n;t)$ for different values of n and at varying t. The function sub-programs generating $I_0(t)$, $I_1(t)$ and $I_n(t)$, $n \geq 2$ are available in Numerical Recipes [37]. So one obvious attempt to generate the filter coefficients $T(n;t)$ is to make use of these function sub-programs. Unfortunately in this approach as the values of t increases arithmetic overflow occurs and the system fails to generate $T(n;t)$ for large t. But a careful observation into the function sub-programs of [37] generating $I_0(t)$ and $I_1(t)$ reveals that both $I_0(t)$ and $I_1(t)$ involve e^t for $t \geq 3.75$. This exponential function is responsible for arithmetic overflow. As t increases e^t increases catastrophically. But we observe that $T(n;t)$ involves e^{-t}. So in order to generate $T(0;t)$ and $T(1;t)$ for $t \geq 3.75$ we omit the term e^t from the function sub-program of $I_0(t)$ and $I_1(t)$ and consequently these function sub-programs generate $T(0;t)$ and $T(1;t)$ for $t \geq 3.75$. To generate $T(0;t)$ and $T(1;t)$ for $0 < t < 3.75$ each of the functions $I_0(t)$ and $I_1(t)$ is multiplied by e^{-t} so that the function sub-programs generating $I_0(t)$ and $I_1(t)$ for $0 < t < 3.75$ return $T(0;t)$ and $T(1;t)$ for $0 < t < 3.75$. The modified function sub-programs generating $T(0;t)$, $T(1;t)$ and $T(n;t)$ for $n \geq 2$ are given in the **Appendix** . These function sub-programs are similar to those available in Numerical Recipes [37] except for the modifications that have been suggested here to avoid arithmetic overflow.

The function sub-program generating $T(n;t)$ involves a parameter "*iacc*". In the function sub-program available in Numerical Recipes [37] this parameter was set to 40. We have changed this value to 400 so as to attain the desired accuracy. Larger values of "*iacc*" were tested but the final results were found to be insensitive to it.

11.2 Truncation of smoothing kernel

A $2-D$ digital curve can be decomposed into two $1-D$ signals namely, x_i denoting the $x-$coordinate and y_i denoting the $y-$coordinate of the ith point of the curve. Each of these $1-D$ signals is convolved with the kernel $T(n;t)$. At the first place we assume that the curve is open and has infinite length. The convolution of x_i and

y_i with $T(n;t)$ is defined by

$$X_i(t) = \sum_{n=-\infty}^{\infty} T(n;t)\, x_{i-n},$$
$$Y_i(t) = \sum_{n=-\infty}^{\infty} T(n;t)\, y_{i-n}. \qquad (11.2)$$

To generate $X_i(t)$ and $Y_i(t)$ the infinite sum on the right hand side of (11.2) should be approximated by a finite one. Following the arguments in [27] a reasonable approach to approximate the infinite sum is to truncate it for some sufficiently large value of n, N (say)

$$X_i(t) = \sum_{n=-N}^{N} T(n;t)\, x_{i-n}$$
$$Y_i(t) = \sum_{n=-N}^{N} T(n;t)\, y_{i-n}, \qquad (11.3)$$

chosen such that the absolute error in $X_i(t)$ and $Y_i(t)$ due to truncation does not exceed a given error limit ϵ_{trunc}. If we assume that both x_i and y_i are bounded functions and

$$max(|\, x_i \,|) = M1,\ max(|\, y_i \,|) = M2\ and\ M = max(M1, M2)$$

then we get the sufficiency condition

$$2M \sum_{n=N+1}^{\infty} T(n;t) \leq \epsilon_{trunc}. \qquad (11.4)$$

So in order to determine the number of filter coefficients that must be generated for a given t to satisfy (11.4) we go on generating the filter coefficients $T(n;t)$ for increasing n. The exact number of filter coefficients required is the maximum n satisfying the condition

$$\sum_{n=-N}^{N} T(n;t) \geq 1 - \epsilon_{trunc}/M. \qquad (11.5)$$

The problem of the choice of ϵ_{trunc} has not been addressed by Lindeberg in [27]. We observe that for a given ϵ_{trunc}, M may be such that ϵ_{trunc}/M falls below $10.0e-6$

and visa-versa. But as seen from the function sub-programs generating $T(0;t)$ and $T(1;t)$ both $I_0(t)$ and $I_1(t)$ are polynomial approximations. The minimum of the absolute error in approximating $I_0(t)$ and $I_1(t)$ is of the order of $10e-7$ [1]. So the sum $\sum_{n=-N}^{N} T(n;t)$ can never be correct beyond the sixth decimal place. So for a given ϵ_{trunc}, M may be such that the desired accuracy may not be attained. We avoid this difficulty by truncating the sum $\sum_{n=-\infty}^{\infty} T(n;t)$ instead of truncating the infinite convolution $X_i(t)$ and $Y_i(t)$ as done in [27]. We generate as many filter coefficients as will satisfy the condition

$$\sum_{n=-N}^{N} T(n;t) \geq 1-\epsilon, \quad \epsilon > 0. \tag{11.6}$$

As seen from this condition the number of filter coefficients to be generated depends on t only but not on the input data as in [27]. For a given t the same number of filter coefficients operate on every curve. We take $\epsilon = 10e-6$ since the maximum of the absolute error in approximating $I_0(t)$ and $I_1(t)$ is of the order of $10e-6$. Larger values of ϵ are feasible from the standpoint of numerical analysis. But it will make the results $var(t) = t$ and $\sum_{-\infty}^{\infty} T(n;t) = 1$ more inaccurate and it may also cause overshoots in the scale-space map. $X_i(t)$ and $Y_i(t)$ are generated using the convolution formulae given in (11.3) and the number of filter coefficients that should be used is determined from the condition (11.6).

11.3 Choice of scale levels

$X_i(t)$ and $Y_i(t)$ are generated for $t > 0$ treating t as a continuous scale parameter. As mentioned in [27] the concept of a continuous scale parameter is of considerable importance, since we are no longer locked to fixed predetermined discrete levels of scale. It allows us to defocus a signal with an arbitrary amount of blurring, which will certainly make it easier to trace the events. But in real implementation it is impracticable to generate the representations at all levels of scale. However, the important idea is that instead of specifying the scale levels in advance, with a continuous scale parameter the scale-space representation at any level of scale can be calculated if desired. This advantage cannot be enjoyed in scale-space analysis with digital Gaussian filter coefficients.

In order to construct the scale-space map the continuous parameter should be sampled at some levels of scale. It is certainly a non-trivial problem to make an appropriate selection of these scale levels. The point of a scale-space having a continuous scale parameter is that it provides a theoretical framework in which the scale steps can be varied arbitrarily.

The problem of choosing scale levels has not been addressed by Lindeberg in [27]. In this section we address the problem of choosing a suitable set of scale levels in real situation. We make the parameter t vary in an interval of 0.01 starting with an initial value of 0.01. The choice of the initial value and the size of the interval is not crucial except for the fact that both of them should be sufficiently small so that t can be treated as a continuous scale parameter. Varying t, the scale-space is sampled at those increasing values of t only for which the maximum number of filter coefficients (N) satisfying the condition (11.6) increases. If $t = t_1$ be the minimum value of t such that the maximum number of filter coefficients satisfying (11.6) is N_1 then to find the next value of t at which the scale-space should be sampled, t is increased from t_1 in an interval of 0.01, the least value of $t(= t_2$, say) for which the maximum number of filter coefficients N_2(say), satisfying the condition (11.6) exceeds N_1, is the next value of t (next to t_1) at which the scale-space should be sampled. So if t_1 and t_2 be two distinct least values of t $(t_2 > t_1)$ for which the maximum number of filter coefficients satisfying the condition (11.6) are N_1 and N_2 respectively then the scale parameter should be sampled at t_1 and t_2 if and only if $N_2 > N_1$. Following this rule the parameter t is sampled from its minimum value to the maximum allowable value.

11.4 Termination criterion

The central problem is the termination criterion. How far t should be varied? At what value of t the smoothing process should stop? What should be the maximum value of t? If N be the maximum number of filter coefficients satisfying the condition (11.6) for a sampled value of t then the window length of the smoothing kernel is $2N + 1$. So for open digital curves t should be increased till the tail of the window does not go beyond the end points of the curve. For a closed digital curve t should

be increased till aliasing effects do not come into play. If a closed digital curve consists of m points then in order to avoid aliasing effects the maximum value of N is determined by

$$\begin{aligned} N_{max} &= \frac{m}{2} - 1, \quad m \text{ even}, \\ &= \frac{m-1}{2}, \quad m \text{ odd}. \end{aligned} \tag{11.7}$$

So t should be varied until N attains N_{max}.

11.5 Construction of scale-space map

To compute curvature from a digital curve we use the first and second order finite differences of the smoothed coordinates $X_i(t)$ and $Y_i(t)$ and the curvature measure as presented in the last chapter following Rattarangsi and Chin [39]. The curvature measure at the ith point and for a given t is given by

$$\kappa_i(t) = \Delta X_i(t)\, \Delta^2 Y_i(t) - \Delta Y_i(t)\, \Delta^2 X_i(t)\,. \tag{11.8}$$

The digital curve is convolved with the kernel $T(n;t)$ and the local extrema of curvature are located at the sampled values of t. The values of i for which $|\kappa_i(t)|$ exceeds $|\kappa_{i-1}(t)|$ and $|\kappa_{i+1}(t)|$ give the location of the extreme curvature points.

The t values are sampled using the increasing N criterion. In order to construct the scale-space map along the $x-$axis (horizontal) the ordinal number i of the points of the curve is shown and along the $y-$axis (vertical) N is shown. The local maxima of absolute curvature at varying N are plotted as points on the xy half plane. The image showing the location of the maxima of absolute curvature at varying N is the scale-space map of the curve. The map consists of an aggregate of dot patterns some of which grow reaching the maximum allowable value of N i.e. N_{max}, some other terminate as N increases. A pair of dot patterns may grow and merge to a single dot pattern which may reach N_{max}. The dot patterns that survive the maximum smoothing are indicative of those maxima that are detected at all levels of detail fine as well coarse. The dot patterns that appear temporarily are indicative of those maxima that are detected at the fine scales but disappear as the degree

of smoothing increases. A number of digital curves and their scale-space map are shown in Figures 11.1 through 11.4.

11.6 Experimental results

The scale-space behavior of corner models such as Γ models, END models and STAIR models when convolved with the discrete scale-space kernel $T(n;t)$ are shown in Figures 11.5 through 11.7. The models being open curves the smoothing should be performed upto a value of t for which N does not go beyond the end points of the model. To detect corners on digital curves the scale-space map of a curve is converted into a tree organization and corners are detected and located in a process of interpreting the tree. The procedure is similar to that presented in the last chapter. Figures 11.1 through 11.4 show the tree organization of the scale-space map of a number of digital curves. The limbs of a tree representing corners are indicated by bold lines. Figures 11.8 through 11.11 show the corners on four digital curves together with the results of the Rattarangsi-Chin algorithm [39]. We observe that the corner detector developed in this chapter detects the least number of corner points on circular objects (please see figure 11.11).

To show the robustness of the corner detector to noise, white Gaussian noise is added to the *digital chromosome* by varying the noise level σ from 0.5 to 2.5 in an interval of 0.5. The noisy curves are shown in Figure 11.12 indicating the corners on them by bold solid circles. From these results we find that as the noise level increases no additional corner is detected. The location of the corner points on the digital curves is comparable to that on the original image.

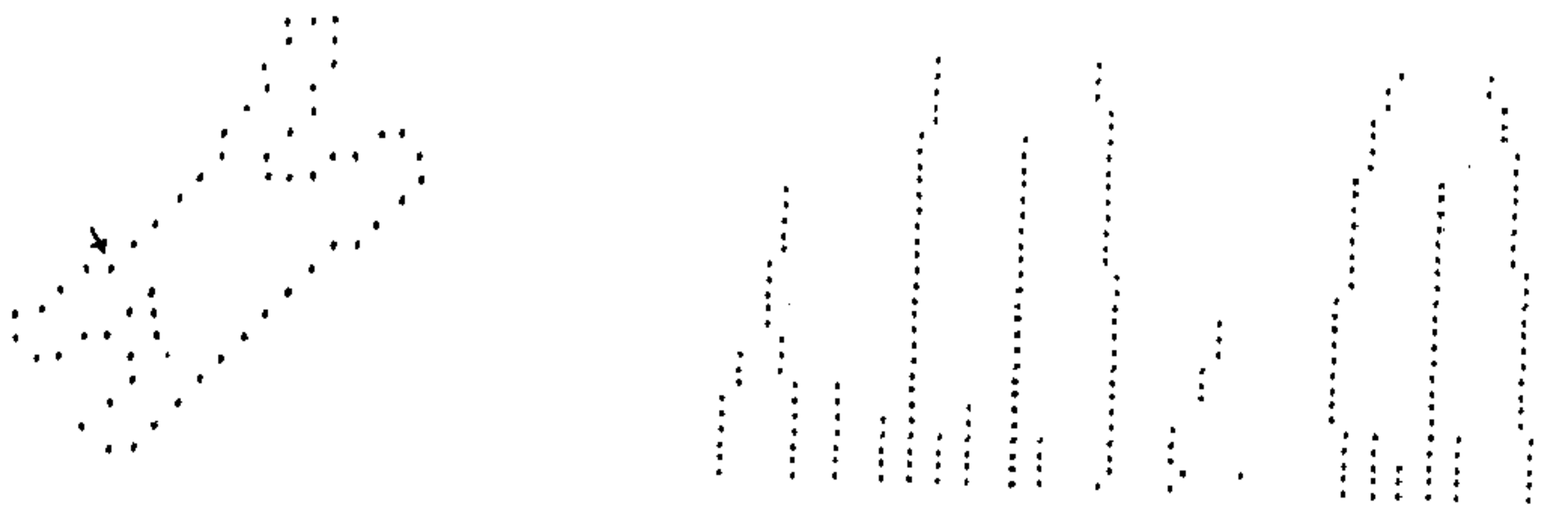


Figure 11.1: The *chromosome* shaped curve (top left), its scale-space map (top right) and tree organization (bottom).

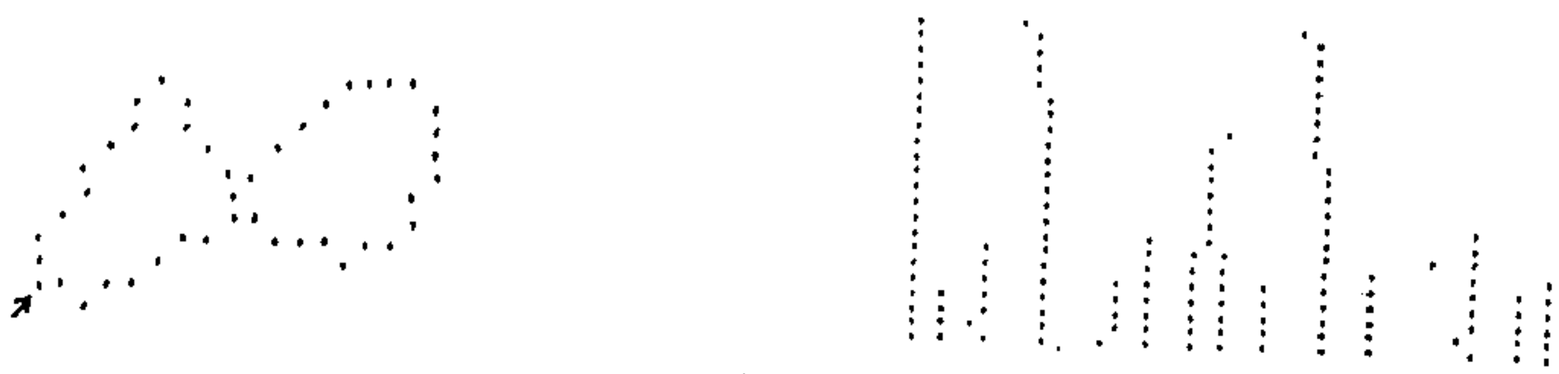


Figure 11.2: The figure-8 curve (top left), its scale-space map (top right) and tree organization (bottom).

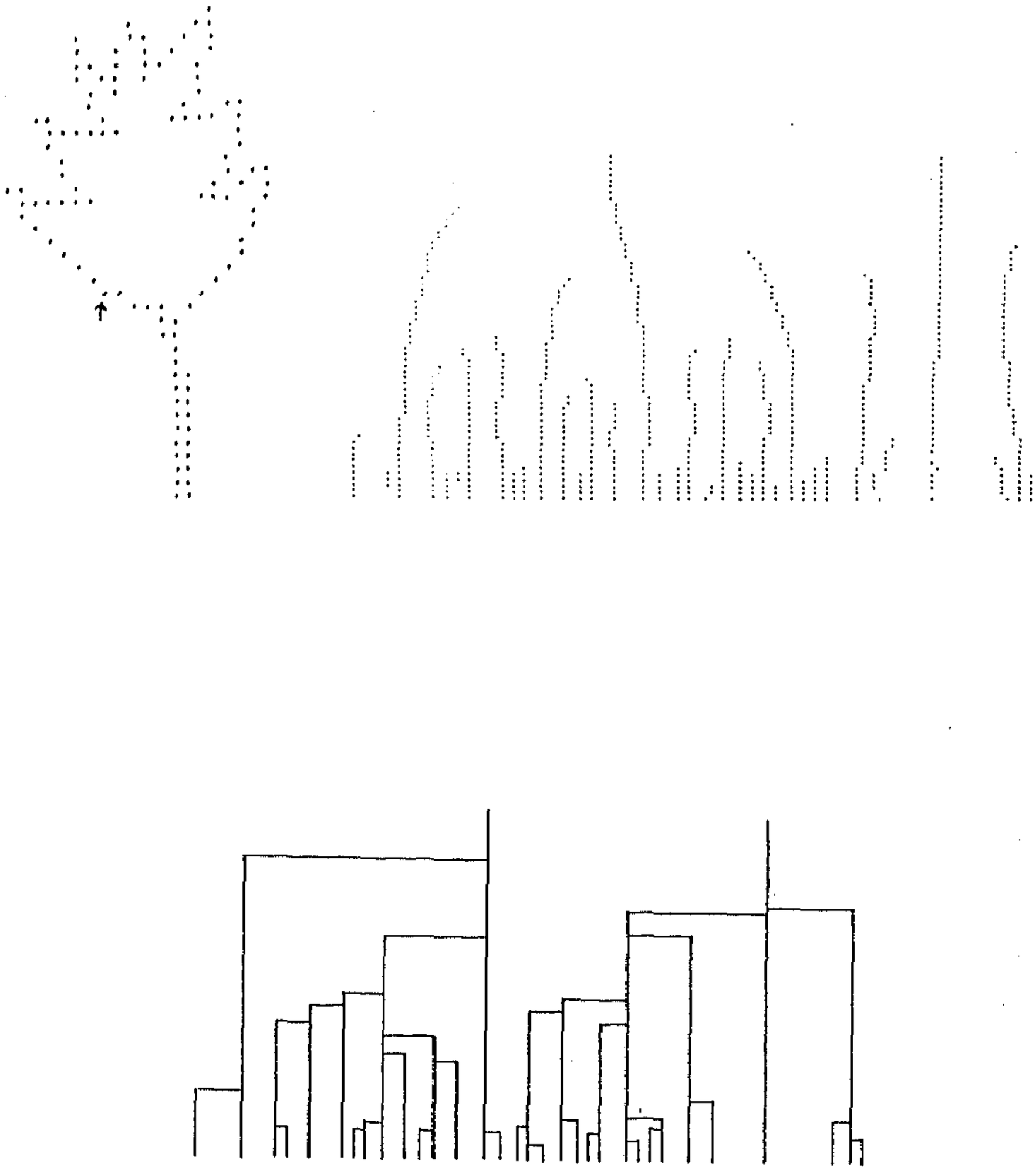


Figure 11.3: The leaf-shaped curve (top left), its scale-space map (top right) and tree organization (bottom).

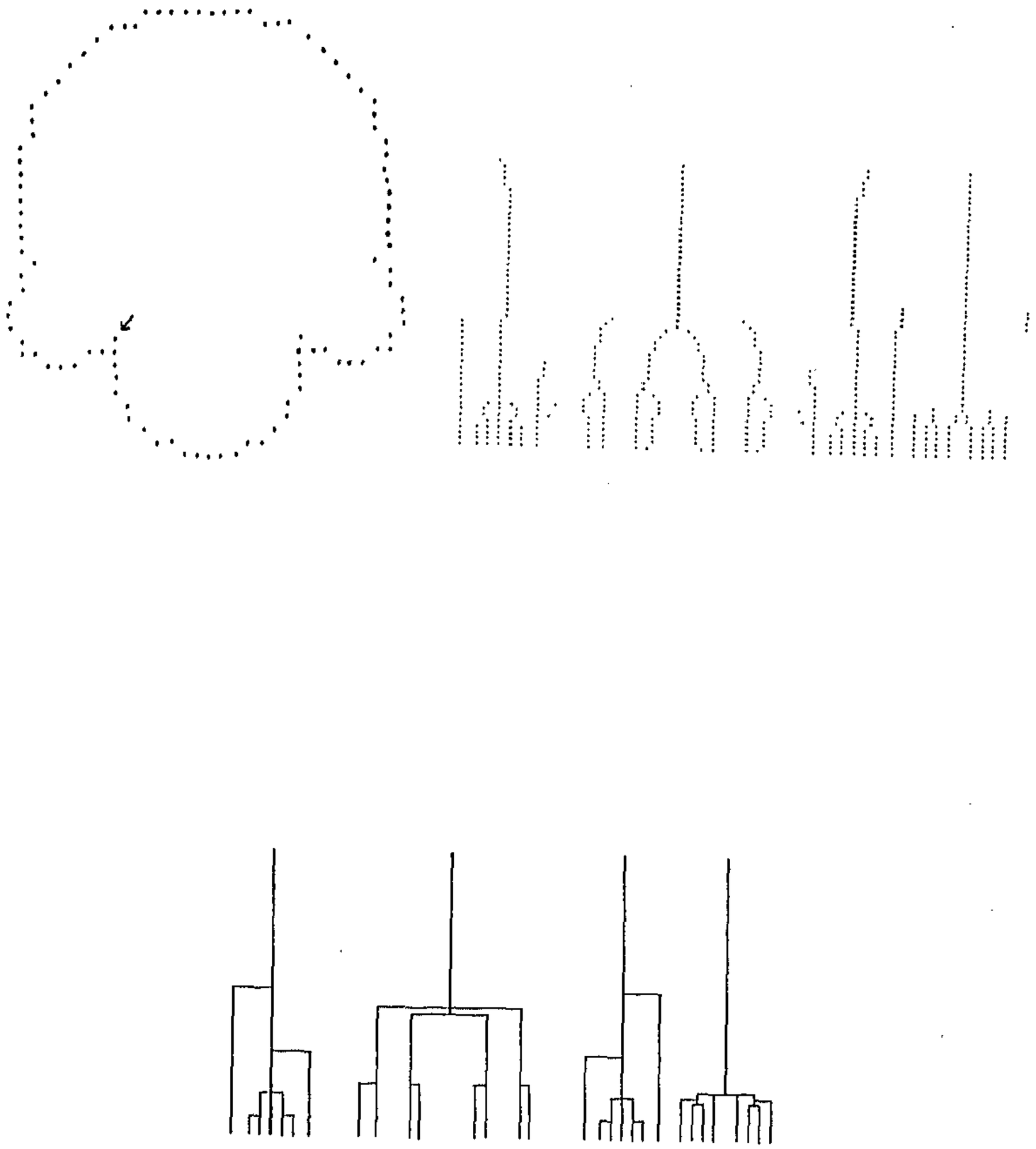


Figure 11.4: The curve with four semi-circles (top left), its scale-space map (top right) and tree organization (bottom).

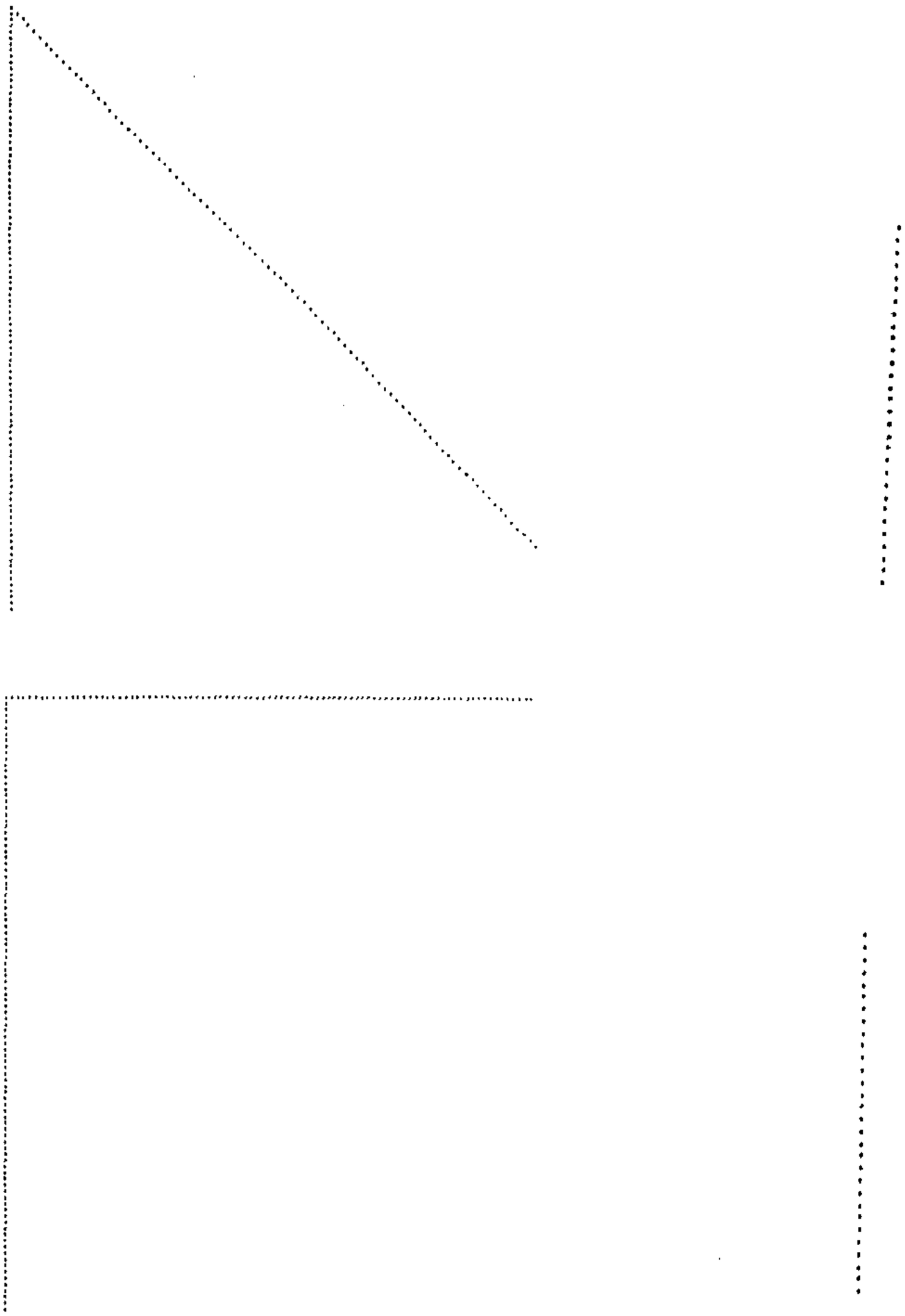


Figure 11.5: Γ models and their scale-space map. The left figure is the model and the right is its scale-space map.

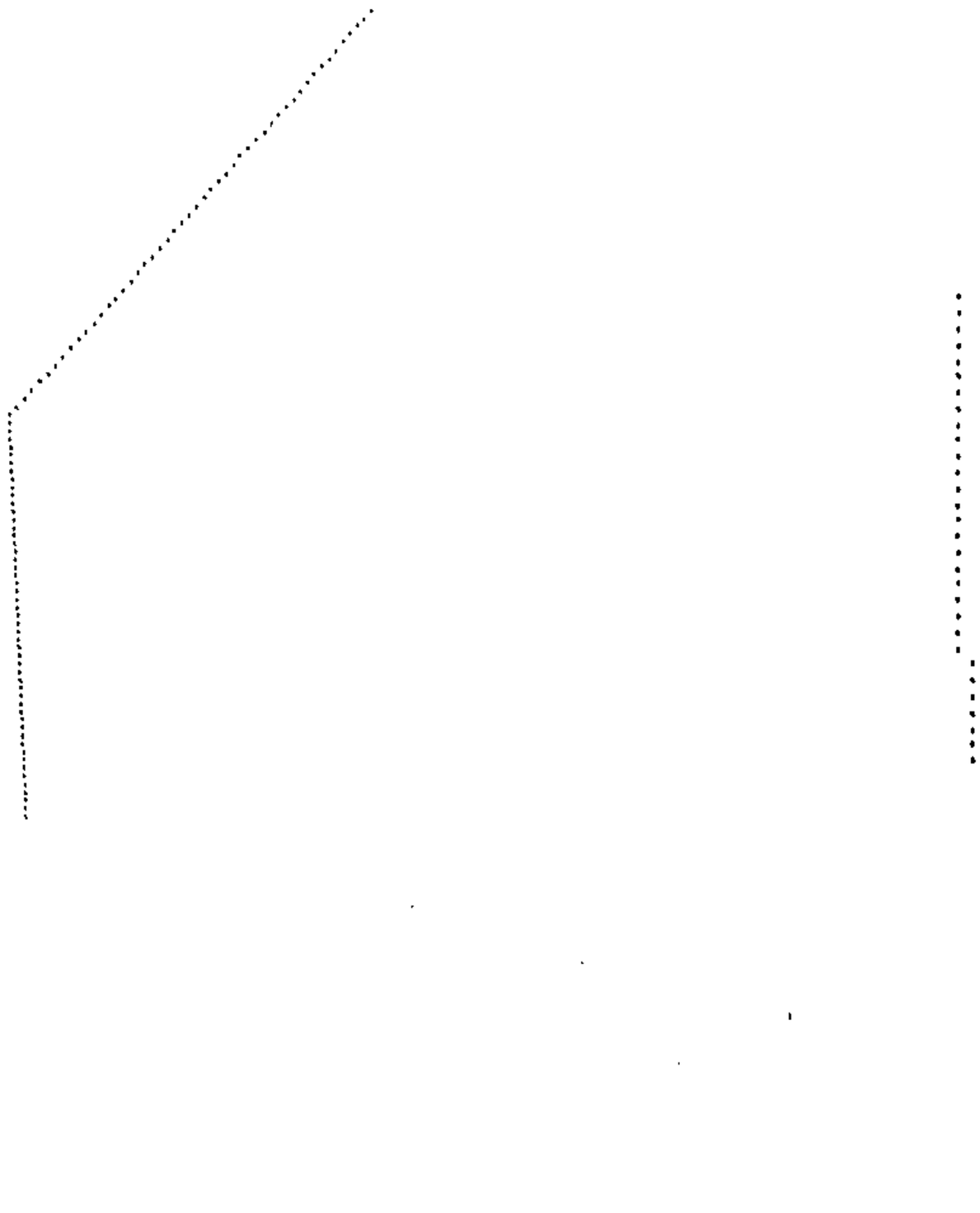
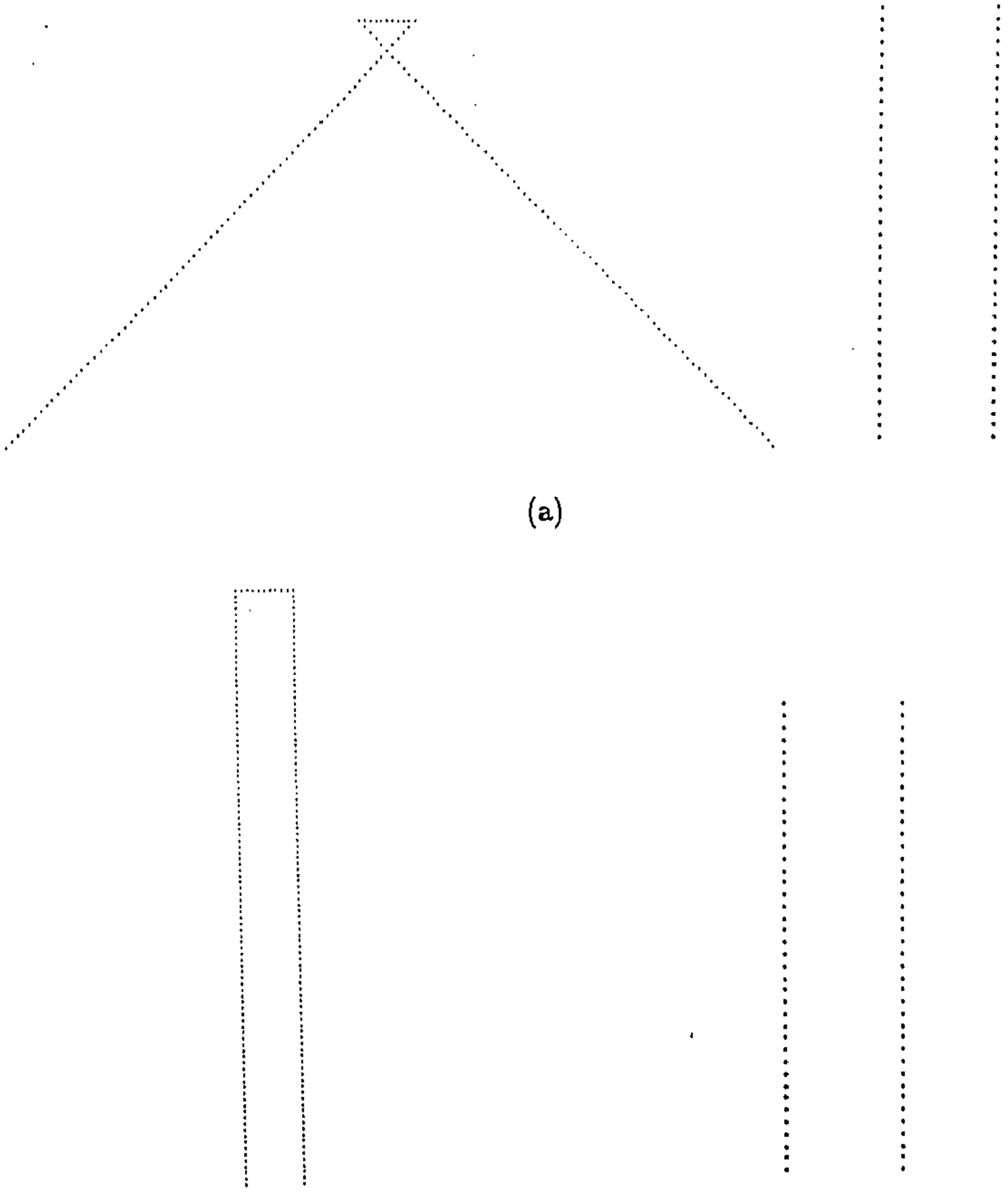


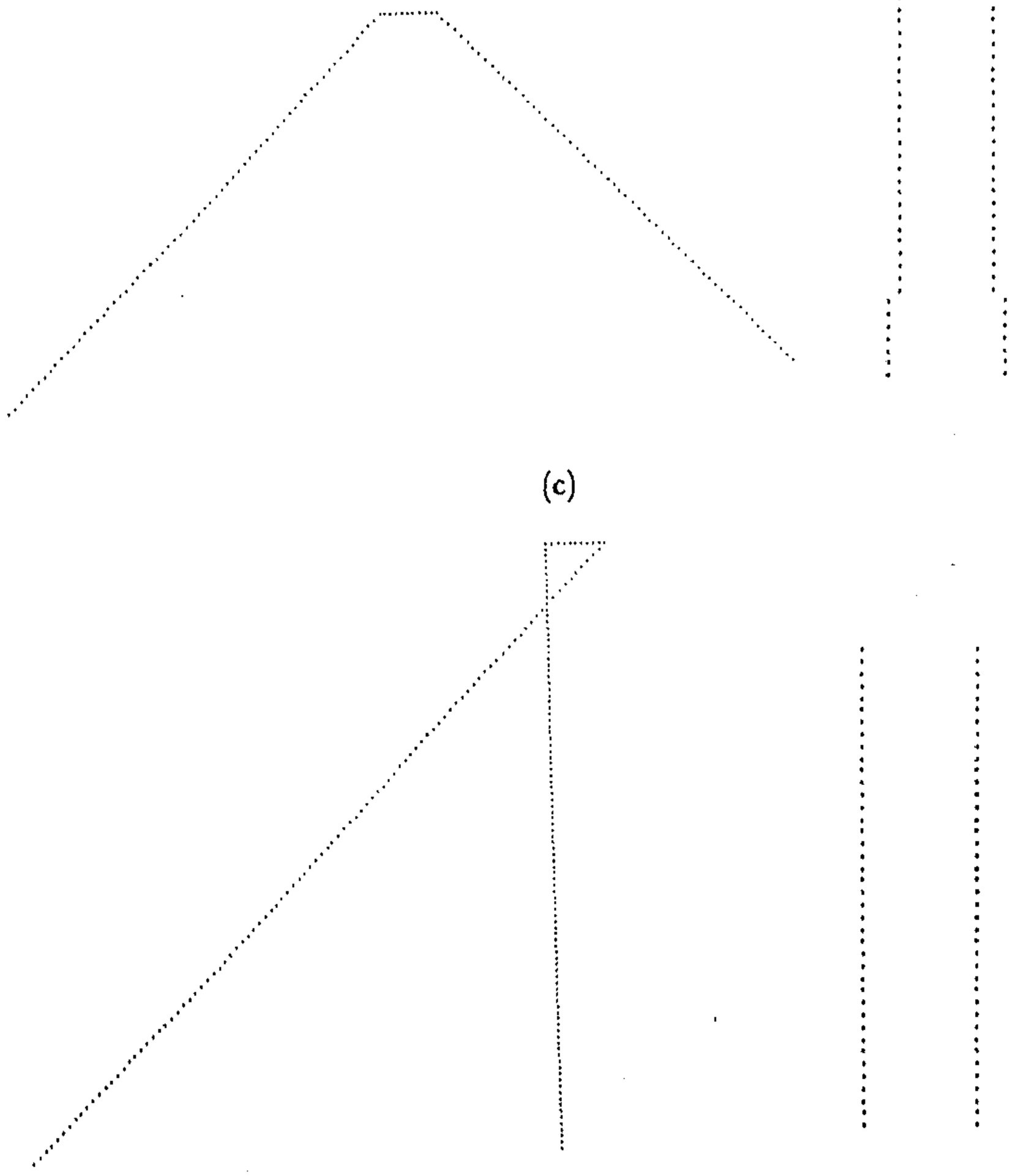
Figure 11.5: Continued.



(a)

(b)

Figure 11.6: END models and their scale-space map. The left figure is the model and the right is its scale-space map.



(c)

(d)

Figure 11.6: Continued.

(e)

(f)

Figure 11.6: Continued.

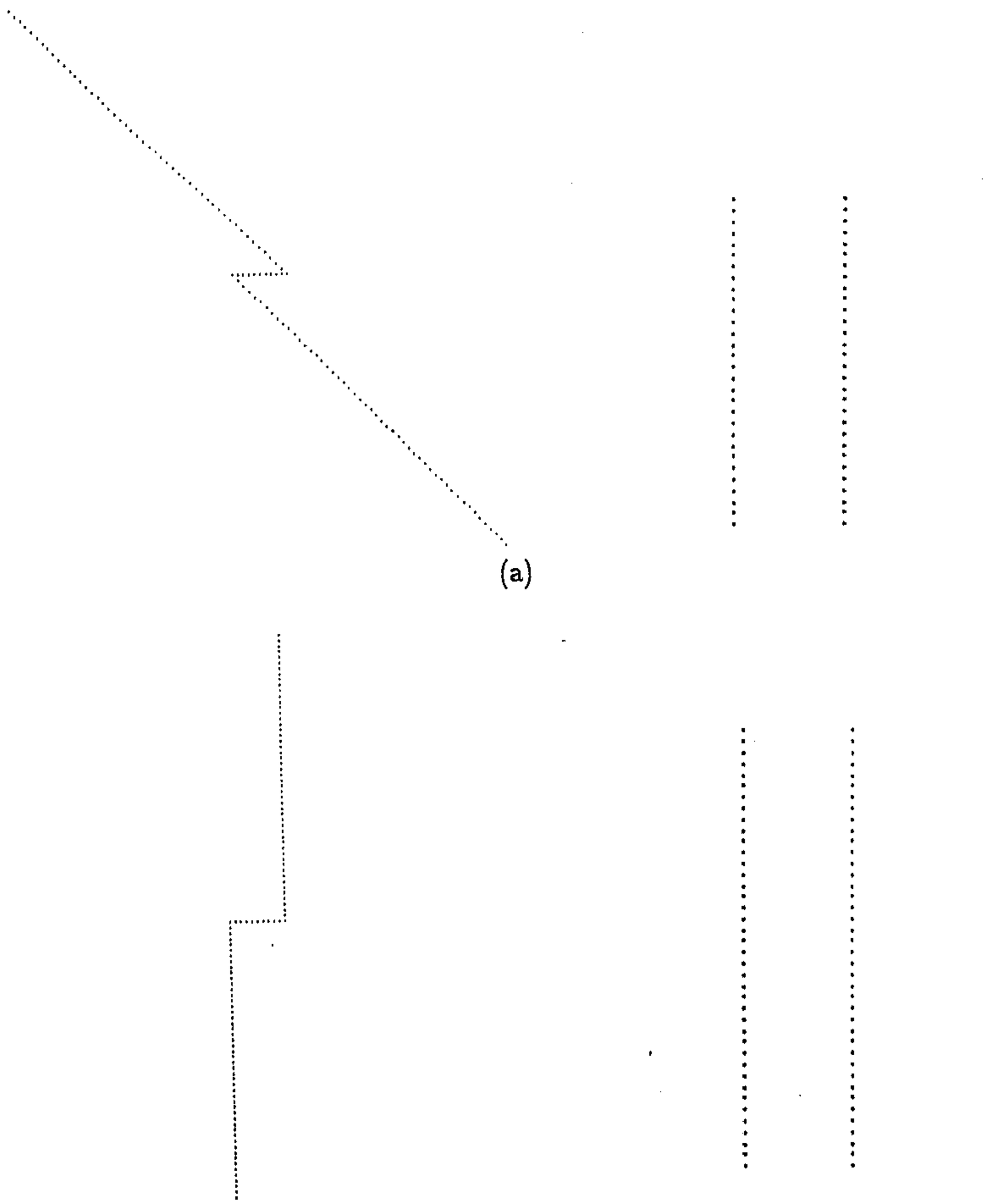
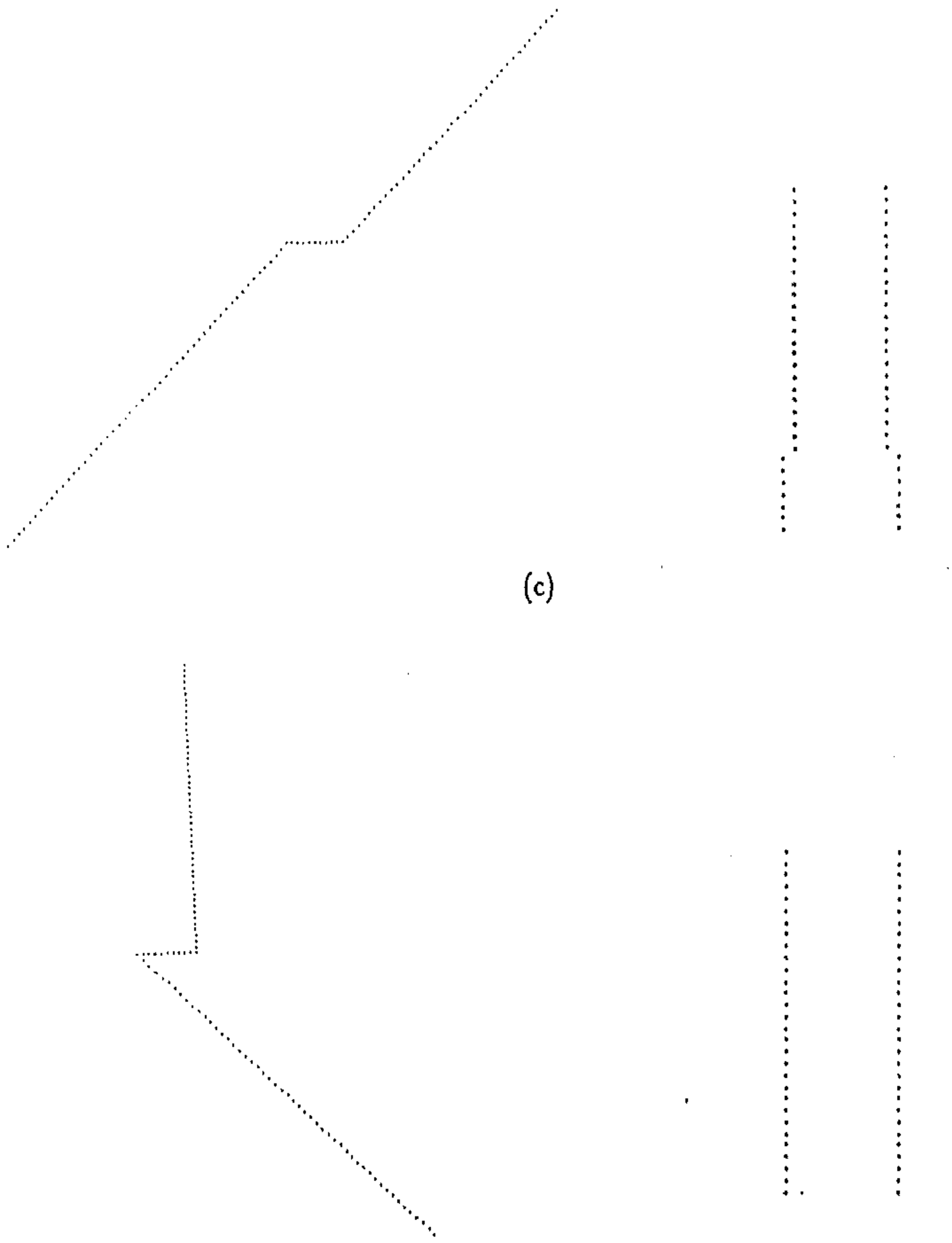


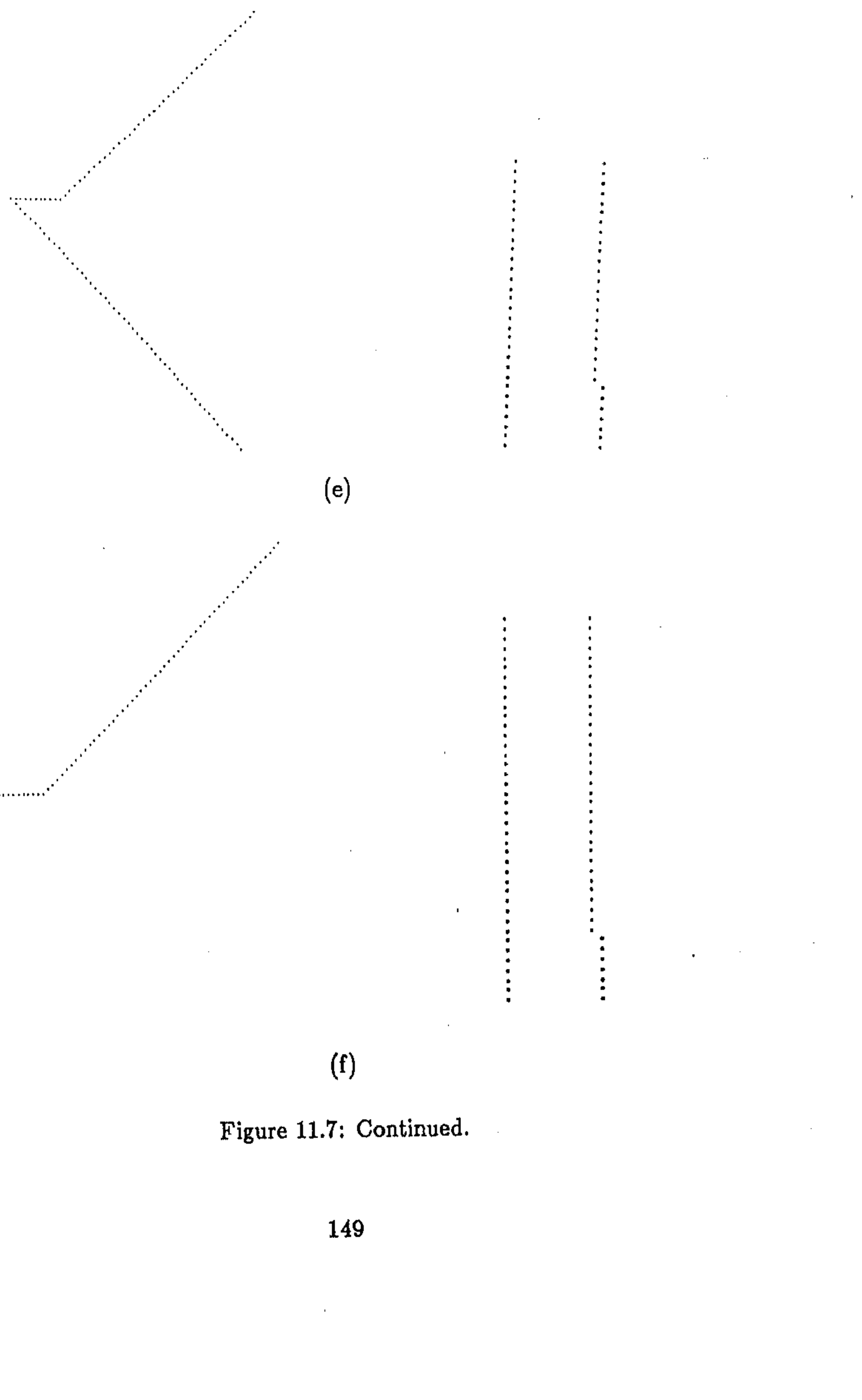
Figure 11.7: STAIR models and their scale-space map. The left figure is the model and the right is its scale-space map.



(c)

(d)

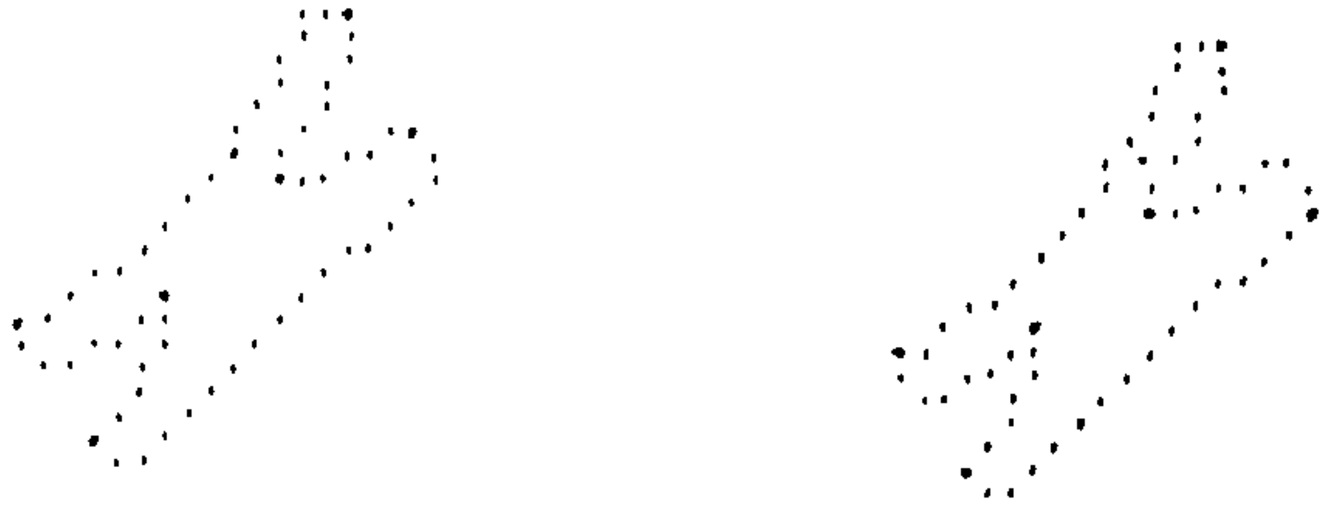
Figure 11.7: Continued.



(e)

(f)

Figure 11.7: Continued.

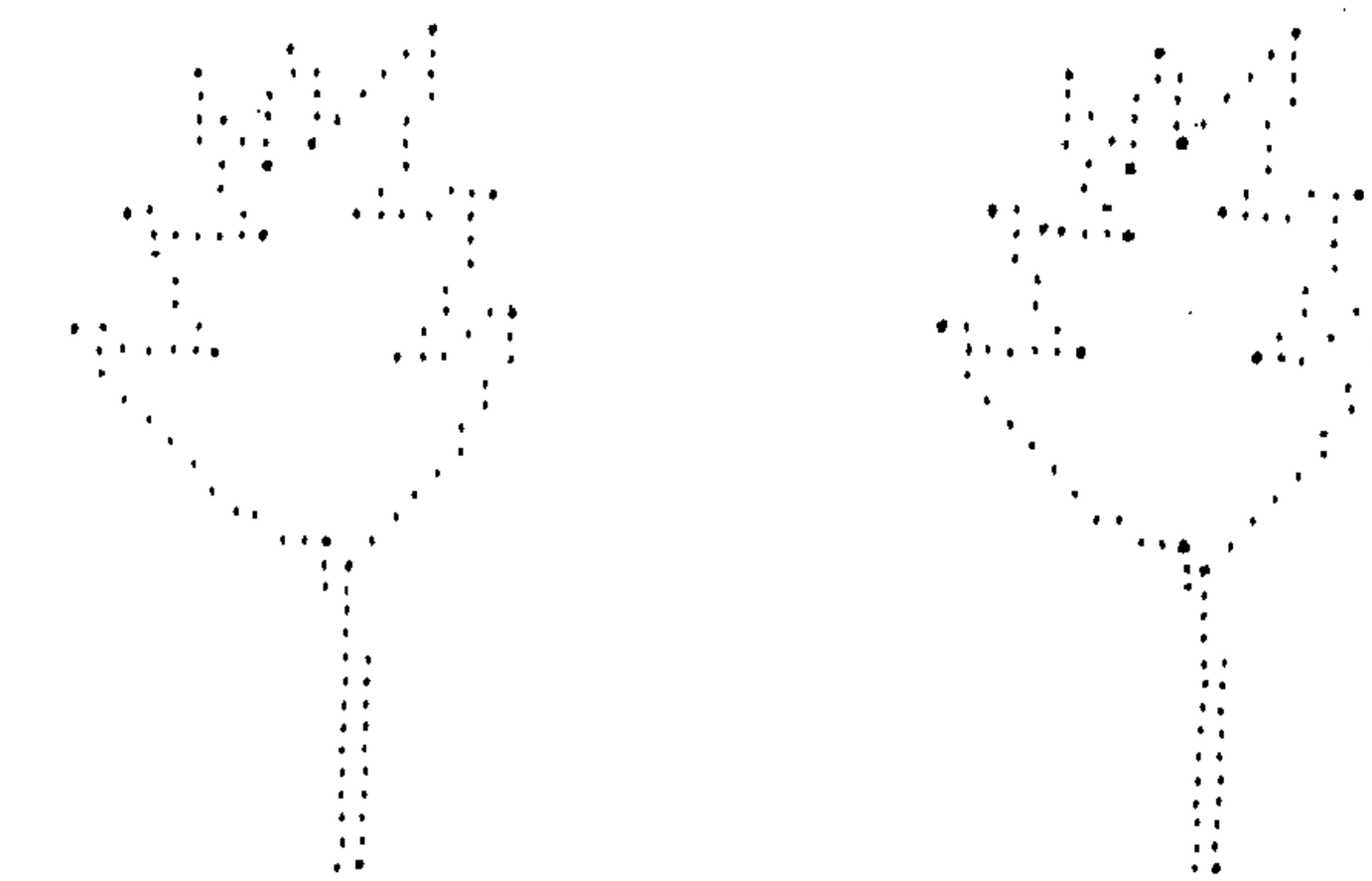


(a) (b)

Figure 11.8: The chromosome shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

(a) (b)

Figure 11.9: The figure-8 curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

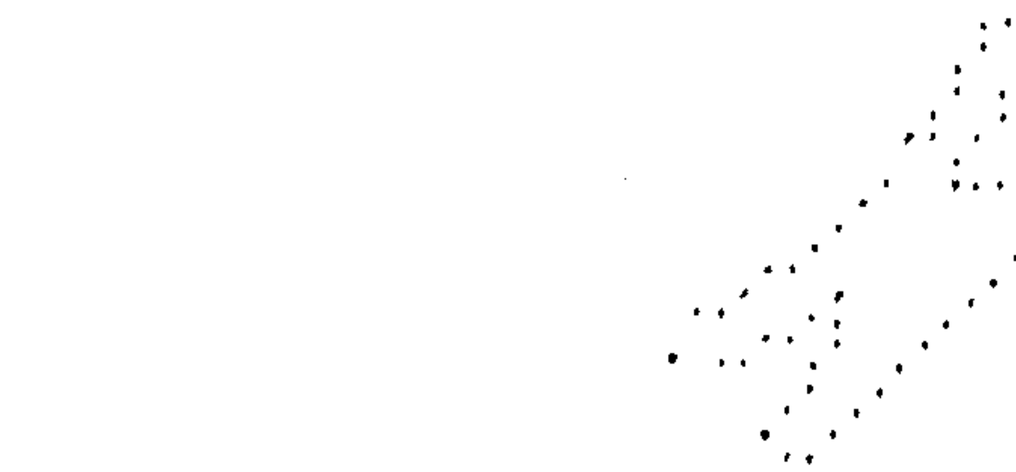


(a) (b)

Figure 11.10: The leaf-shaped curve. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.

(a) (b)

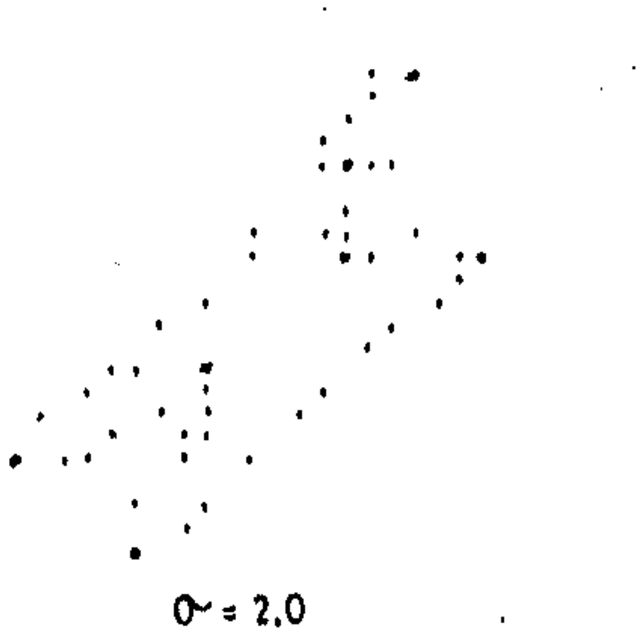
Figure 11.11: The curve with four semi-circles. The corners are indicated by bold solid circles. (a) Present method, (b) Rattarangsi-Chin method.



$\sigma = 0.5$

$\sigma = 1.0$

$\sigma = 1.5$



$\sigma = 2.0$

$\sigma = 2.5$

Figure 11.12: The noisy chromosome at varying noise levels. The corners are indicated by bold solid circles.

Chapter 12

Conclusion

We have developed a series of algorithms for polygonal approximation of closed digital curves. The problem has been treated as a side detection as well as an angle detection problem.

The split-and-merge technique introduced in chapter 2 resolves the problem of initial segmentation. The initial segmentation technique is simpler than that of the Ansari-Delp algorithm [3] as it requires less number of arithmetic operations than that of the Ansari-Delp algorithm. The procedure has been tested with a number of digital curves. The experimental results have been compared with the Ansari-Delp algorithm. The procedure has been found to produce more accurate results than the Ansari-Delp algorithm.

From chapter 3 through chapter 6 we have developed a series of sequential algorithms. The algorithm developed in chapter 3 is a one-pass algorithm which is simple and fast. It involves no arithmetics except subtractions. The approximation error is controlled indirectly by the counter. The higher the counter is, the higher is the approximation error. The procedure is purely a searching technique which is done in a sequential fashion. Though this algorithm is simple and fast but it holds for curves with uniformly spaced points only. It may be possible to make this procedure applicable to curves with non-uniformly spaced points also by replacing the *finite differences* by *divided differences*.

The algorithm developed in chapter 4 is sequential one-pass and holds for curves with uniformly spaced points as well as non-uniformly spaced points. This algorithm

is more efficient than the other sequential one-pass algorithms such as Williams [59], Wall and Danielsson [58] in the sense that it involves less number of arithmetic operations than the others. It need a small and finite memory and detects the sides on fly. The approximation error is controlled by the critical values of the area and perimeter of triangles. The approximation made in computing the side length of the triangles is taken from Wall and Danielsson [58] and has also been used in many problems of image processing [22]. We have used this approximation because it produces good results reducing the computational load significantly. One can as well use Tchebycheff's approximation at the cost of higher computational load.

In each of these algorithms it is necessary to specify the maximum allowable error. In the algorithms developed in chapters 3 and chapter 4 the maximum allowable error is specified indirectly by the counter and the critical values of the area and the perimeter of the triangles. In the other existing algorithms too, either the number of line segments or the maximum allowable error is specified (either directly or indirectly).

In chapter 5 and chapter 6 we have shown how to make a polygonal approximation without specifying the maximum allowable error or the number line segments. In these algorithms we keep both the number of line segments and the maximum allowable error unspecified and allow the procedure to determine it on the basis of the local topography of the curve. So these procedures can run without operator's intervention.

In chapter 5 integral square error has been used to measure the closeness of the polygon to the curve. In chapter 6 we have shown that though the most commonly used norms are integral square error and the maximum error, but it is also possible to use the sum of absolute errors (L_1 norm) as a measure of closeness.

Both these algorithms are sequential one-pass and they hold for curves with uniformly spaced points as well as non-uniformly spaced points. Unlike the existing sequential algorithms, neither do they round off sharp turnings nor do they miss corners.

Each of the sequential algorithms that we have developed has been compared with the Williams' [59] and the Wall-Danielsson [58] algorithm both of which are also sequential and one-pass in nature. In each case our algorithms produce more

accurate results than either of the two algorithms. Our procedures neither miss corners nor do they round off sharp turnings. The Williams' algorithm misses corners and rounds off sharp turnings. The Wall-Danielsson algorithm does not round off sharp turnings but sometimes it detects false vertices. The peak-test introduced by Wall and Danielsson succeeds to retain the sharp turnings but it fails to locate a vertex at its actual position when the turning is not so sharp i.e. where the length of the line joining the last vertex to the current point exceeds that joining the last vertex to the last point that passed the test.

In chapter 7 and chapter 8 polygonal approximation has been treated as an angle detection problem. In contrast to the other algorithms, here all processing are done locally and hence they are suitable for parallel processing. In chapter 7 we have shown that it is possible to use k-cosine to determine the region of support without using any input parameter. A new measure of significance named smoothed k-cosine has been introduced. This measure of significance is different from the smoothed k-cosine introduced by Rosenfeld and Weszka [50]. The procedure has been applied on a number of digital curves and promising results have been obtained. It is found to perform well on curves which consist of features of multiple size.

In chapter 8 we have introduced the concept of asymmetric region of support and a new measure of significance named $k - l$ cosine. The procedure has been tested with a number of digital curves and promising results have been obtained. The procedure does not need any input parameter and performs well on curves which consist of features of multiple size.

Each of these algorithms have been compared with the Teh and Chin algorithm [56] and in each case our procedures are found to detect more significant / dominant points than the Teh-Chin algorithm.

In chapter 9 we have made scale-space analysis using one of our polygonal approximation schemes and have shown its application to corner detection. Our procedure is shown to detect more corner points than the Rattarangsi-Chin algorithm [39]. This scale-space analysis, in contrast to the existing scale-space technique does not require convolution of the curve with a smoothing kernel. It does not detect corners via curvature estimation. It holds for curves with uniformly spaced points only.

As the conventional scale-space analysis technique involves convolution of a curve with a smoothing kernel, in chapter 10 we have made scale-space analysis by repeated convolution of a curve using Gaussian kernel with constant window size. This is in contrast to the existing Gaussian smoothing process which makes use of the Gaussian kernel with varying window size. This technique reduces the space requirements to a small and finite quantity from $O(n^2)$ which is required in the existing Gaussian smoothing process. The computational load has also been shown to be reduced to $O(n)$ from $O(n^2)$ in the existing Gaussian smoothing process. The smoothing process has been shown to enjoy scale-space property. The scale-space map has been used to detect corners on digital curves. The corner detector has been found to produce more corner points than the Rattarangsi-Chin algorithm [39]. The corner detector has also been found to be robust to noise.

In chapter 11 we have used the discrete scale-space kernel with a continuous scale parameter proposed by Lindeberg [27] to make scale-space analysis and corner detection on $2-D$ digital curves. The numerical problems that arise in the implementation of the discrete kernel have been addressed and possible solutions are proposed. The corner detector has been found to produce the least number of corners on curves that consist of circular arcs (Figure 11.11). The corner detector has also been found to be robust to noise.

From the last three chapters on scale-space analysis we conclude that scale-space analysis of digital curves without convolution with a smoothing kernel is feasible provided the smoothing technique preserves scale-space property. Scale-space analysis of discrete curves using a continuous scale-space kernel is also feasible provided the continuous kernel is discretised properly. Scale-space analysis of discrete curves using discrete scale-space kernel with a continuous scale parameter is also feasible with some suitable assumptions.

Bibliography

[1] Abramonowitz, M. and I. A. Stegun (1964). Handbook of mathematical functions (Applied mathematics series, 55), Wat. Bureau Standards.

[2] Anderson, I. M. and J. C. Bezdek (1984). Curvature and tangential deflection of discrete arcs: A theory based on the commutor of scatter matrix pair and its application to vertex detection in plane shape data. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-6, pp. 27-40.

[3] Ansari, N. and Edward J. Delp (1991). On detecting dominant points. Pattern Recognition, vol. 24, pp. 441-550.

[4] Aoyama, H. and M. Kawagoe (1991). A piecewise linear approximation method preserving visual feature points of original figures. CVGIP: Graphical Models and Image Processing, vol. 53, pp. 435-446.

[5] Asada, H. and M. Brady (1986). The curvature primal sketch. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 2-14.

[6] Attneave, F. (1954). Some informational aspects of visual perception. Psychological Review, vol. 61, pp. 183-193.

[7] Babaud,J., A. P. Witkin, M. Baudin and R. O. Duda (1986). Uniqueness of Gaussian kernel for scale-space filter. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 26-33.

[8] Bellman, R. (1961). Dynamic Programming, Princeton, N.J. Princeton University Press.

[9] Bellman, R. (1961). On the approximation of curves by line segments using dynamic programming. Communications of ACM, vol. 4, pp. 284.

[10] Burt, P. J. (1981). Fast filter transforms for image processing. Computer vision, graphics and image processing, vol. 16, pp. 20-51.

[11] Burt, P. J. and E. H. Adelson (1983). The Laplacian pyramid as a compact image code. IEEE Transactions on Communications, vol. COM-31, no. 4.

[12] Cantoni, A. (1971). Optimal curve fitting with piecewise polynomials. IEEE Transactions on Computers, vol. C-26, pp. 59-67.

[13] Duda, R. O. and P. E. Hart (1973). Pattern Classification and Scene Analysis, Wiley, New York, pp. 328-329.

[14] Dunham, J. G. (1986). Optimum uniform piecewise linear approximation of planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 67-78.

[15] Davis, L. S. (1977). Understanding shape: angles and sides. IEEE Transactions on Computers, vol. C-26, pp. 236-242.

[16] Fahn, C. S., J. F. Wang and J. Y. Lee (1989). An adaptive reduction procedure for piecewise linear approximation of digitized curves. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-11, pp. 967-973.

[17] Freeman, H. and L. S. Davis (1977). A corner-finding algorithm for chain-coded curves. IEEE Transactions on Computers, vol. C-26, pp. 287-303.

[18] Freeman, H. (1967). On the classification of line drawing data. In:W. Wather Dunn. Ed., Models for perception of speech and visual form; MIT Press, Cambridge, MA, pp. 408-412.

[19] Gluss, Brian (1962). Further remarks on line segment curve fitting using dynamic programming. Communications of ACM, vol. 5, pp. 441-443.

[20] Gluss, Brian (1962). A line segment curve fitting algorithm related to optimal encoding of information. Information and Control, vol. 5, pp. 261-267.

[21] Gluss, Brian (1964). An alternative method for continuous line segment curve fitting. Information and Control, vol. 7, pp. 200-206.

[22] Gonzalez, R. C. and P. Wintz (1987). Digital Image Processing. Addision-Wesley, Reading, MA.

[23] Gupta, L. and K. Malakapalli (1990). Robust partial shape classification using invariant breakpoints and dynamic alignment. Pattern Recognition, vol. 23, pp. 1103-1111.

[24] Imai, H. and M. Iri (1986). Computational geometric methods for polygonal approximations of a curve. Computer Vision, Graphics and Image Processing, vol. 36, pp. 31-41.

[25] Koenderink, J. J. and A. J. van Doorn (1984). The structure of images. Biological Cybernetics, vol. 50, pp. 363-370.

[26] Kurozumi, Y. and W. A. Davis (1982). Polygonal approximation by the minimax method. Computer Graphics and Image Processing, vol. 19, pp. 248-264.

[27] Lindeberg, T. (1990). Scale-space for discrete signals. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-12, pp. 234-254.

[28] Medioni, G. and Y, Yasumoto (1987). Corner detection and curve representation using cubic B-splines. Computer Vision, Graphics and Image Processing, vol. 31, pp. 267-278.

[29] Meer, P., E. S. Baugher and A. Rosenfeld (1987). Frequency domain analysis and synthesis of image pyramid generating kernels. IEEE Transactions on Pattern analysis and Machine Intelligence, vol. PAMI-9, no.4, pp. 512-522.

[30] Meer, P., E. S. Baugher and A. Rosenfeld (1988). Extraction of trend lines and extrema from multiscale curves. Pattern Recognition, vol. 21, pp. 217-226.

[31] Mokhtarian, F. and A. Mackworth (1986). Scale-based description and recognition of planar curves and two-dimensional shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, pp. 34-43.

[32] Pavlidis, T. (1982). Algorithms for Graphics and Image Processing, Computer Science Press, New York, pp. 281-297.

[33] Pavlidis, T. and S. L. Horowitz (1974). Segmentation of plane curves. IEEE Transactions on Computers, vol. C-23, pp. 860-870.

[34] Pavlidis, T. (1977). Polygonal approximation by Newton's method. IEEE Transactions on Computers, vol. C-26, pp. 800-807.

[35] Pavlidis, T. (1973). Waveform segmentation through functional approximation. IEEE Transactions on Computers, vol. C-22, pp. 689-697.

[36] Pe'rez, J.-C. and E. Vidal (1992). An algorithm for optimum piecewise linear approximation of digitized curves. In Proc. 11th IAPR International Conference on Pattern Recognition, The Hague, The Netherlands, Aug. 30 - Sep. 3, vol. 3, pp. 167-170.

[37] Press, W. H., B. P. Flannery, S. A. Teukolsky and W. T. Vetterling (1986). Numerical Recipes: The art of computer programming. London: Cambridge University Press.

[38] Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing, vol. 1, pp. 244-256.

[39] Rattarangsi, A. and R. T. Chin (1992). Scale-based detection of corners of planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-14, pp. 430-449.

[40] Ray, B. K. and K. S. Ray (1991). A new approach to polygonal approximation. Pattern Recognition Letters, vol. 12, pp. 229-234.

[41] Ray, B. K. and K. S. Ray (1992). An algorithm for polygonal approximation of digitized curves. Pattern Recognition Letters, vol. 13, pp. 489-496.

[42] Ray, B. K. and K. S. Ray (1994). A non-parametric sequential method for polygonal approximation of digital curves. Pattern Recognition Letters, vol. 15, pp. 161-167.

[43] Ray, B. K. and K. S. Ray (1993). An optimal algorithm for polygonal approximation of digital curves using L_1 norm. Pattern Recognition, vol. 26, pp. 505-509.

[44] Ray, B. K. and K. S. Ray (1992). Detection of significant points and polygonal approximation of digitized curves. Pattern Recognition Letters, vol. 13, pp. 443-452.

[45] Ray, B. K. and K. S. Ray (1992). An algorithm for detection of dominant points and polygonal approximation of digitized curves. Pattern Recognition Letters, vol. 13, pp. 849-856.

[46] Ray, B. K. and K. S. Ray (1994). A new approach to scale-space image. Pattern Recognition Letters, vol. 15, pp. 365-372.

[47] Roberge, J. (1985). A data reduction algorithm for planar curves. Computer Vision, Graphics and Image Processing, vol. 29, pp. 168-195.

[48] Rosenfeld, A. and E. Johnston (1973). Angle detection on digital curves. IEEE Transactions on Computers, vol. C-22, pp. 875-878.

[49] Rosenfeld, A. and M. Thurston (1971). Edge and curve detection for digital scene analysis. IEEE Transactions on Computers, vol. C-20, pp. 562-569.

[50] Rosenfeld, A. and J. S. Weszka (1975). An improved method of angle detection on digital curves. IEEE Transactions on Computers, vol. C-24, pp. 940-941.

[51] Scarborough, James B. (1966). Numerical Mathematical Analysis, Oxford and IBH Publishing Company.

[52] Saint-Marc, P., Jer-Sen Chen and G. Medioni (1991). Adaptive Smoothing: A general tool for early vision. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-13, pp. 514-529.

[53] Sankar, P. V. and C. V. Sharma (1978). A parallel procedure for the detection of dominant points on a digital curve. Computer Graphics and Image Processing, vol. 7, pp. 403-412.

[54] Skalansky, J. and V. Gonzalez (1980). Fast polygonal approximation of digitized curves. Pattern Recognition, vol. 12, pp. 327-331.

[55] Stone, H. (1961). Approximation of curves by line segments. Mathematics of Computation, vol. 15, pp. 40-47.

[56] Teh, C. H. and R. T. Chin (1989). On the detection of dominant points on digital curves. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-11, pp. 859-872.

[57] Tomek, I. (1974). Two algorithms for piecewise linear approximation for functions of one variable. IEEE Transactions on Computers, vol. C-23, pp. 445-448.

[58] Wall, K. and P.-E. Danielsson (1984). A fast sequential method for polygonal approximation of digitized curves. Computer Vision, Graphics and Image Processing, vol. 28, pp. 220-227.

[59] Williams, C. M. (1978). An efficient algorithm for the piecewise linear approximation of planar curves. Computer Graphics and Image Processing, vol. 8, pp. 286-293.

[60] Williams, C. M. (1981). Bounded straight line approximation of digitized planar curves and lines. Computer Vision, Graphics and Image Processing, vol. 16, pp. 370-381.

[61] Witkin, A. P. (1983). Scale-space filtering. Proceedings of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, Germany, pp. 1019-1021.

[62] Wu, J.-S. and J.-J. Leou (1993). New polygonal approximation schemes for object shape recognition. Pattern Recognition, vol. 26, pp. 471-484.

[63] Wu, L.-D. (1984). A piecewise linear approximation based on a statistical model. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-6, pp. 41-45.

[64] Yuilli, A. L. and T. A. Poggio (1986). Scaling theorems for zero-crossings. IEEE transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 1, pp. 15-25.

Appendix
Function sub-programs generating $T(n;t)$

```
      function bessio (x)
c     returns the modified bessel function Io(x) for all real x
      real p1,p2,p3,p4,p5,p6,p7,q1,q2,q3,q4,q5,q6,q7,q8,q9
      data p1,p2,p3,p4,p5,p6,p7/1.0d0,3.5156229d0,3.0899424d0,
     c 1.2067492d0,0.2659732d0,0.360768d-1,0.45813d-2/
      data q1,q2,q3,q4,q5,q6,q7,q8,q9/0.39894228d0,0.1328592d-1,
     c 0.225319d-2,-0.157565d-2,0.916281d-2,-0.2057706d-1,0.2635537d-1,
     c -0.1647633d-1,0.392377d-2/
      if(abs(x) .lt. 3.75) then
      y=(x/3.75)**2
      bessio=exp(-x)*(p1+y*(p2+y*(p3+y*(p4+y*(p5+
     c y*(p6+y*p7))))))
      else
      ax=abs(x)
      y=3.75/ax
      bessio1=1.0/sqrt(ax)
      bessio2=(q1+y*(q2+y*(q3+y*(q4+y*(q5+y*(q6+y*(q7+
     c y*(q8+y*q9)))))))
      bessio=bessio1*bessio2
      endif
      return
      end

      function bessi1(x)
c     returns the modified bessel function I1(x) for real x
      real p1,p2,p3,p4,p5,p6,p7,q1,q2,q3,q4,q5,q6,q7,q8,q9
      data p1,p2,p3,p4,p5,p6,p7/0.5d0,0.87890594d0,0.51498869d0,
     c 0.15084934d0,0.2658733d-1,0.301532d-2,0.32411d-3/
      data q1,q2,q3,q4,q5,q6,q7,q8,q9/0.39894228d0,-0.3988024d-1,
```

```
c       -0.362018d-2,0.163801d-2,-0.1031555d-1,0.22829670d-1,
c       -0.2895312d-1,0.1787654d-1,-0.420059d-2/
        if(abs(x) .lt. 3.75) then
        y=(x/3.75)**2
        bessi1=exp(-x)*x*(p1+y*(p2+y*(p3+y*(p4+
c       y*(p5+y*(p6+y*p7))))))
        else
        ax=abs(x)
        y=3.75/ax
        u=1.0/sqrt(ax)
        v=q1+y*(q2+y*(q3+y*(q4+y*(q5+y*(q6+y*(q7+
c       y*(q8+y*q9)))))))
        bessi1=u*v
        if(x .lt. 0.0) bessi1=-bessi1
        endif
        return
        end

        function bessi(n,x)
c       returns the modified bessel function of integer order
        parameter( iacc=400 , bigno=1.0e10 , bigni=1.0e-10 )
        if(n.lt.2) pause 'bad argument n in bessi'
        if(x.eq.0.0) then
        bessi=0.0
        else
        tox=2.0/abs(x)
        bip=0.0
        bi=1.0
        bessi=0.0
        m=2*(n+int(sqrt(float(iacc*n))))
        do 10 j=m,1,-1
        bim=bip+float(j)*tox*bi
```

```
      bip=bi
      bi=bim
      if (abs(bi).gt.bigno) then
      bessi=bessi*bigni
      bi=bi*bigni
      bip=bip*bigni
      endif
      if(j.eq.n) bessi=bip
10    continue
      bessi=bessi*bessio(x)/bi
      if(x .lt. 0.0 .and. mod(n,2) .eq. 1) bessi=-bessi
      endif
      return
      end
```