1986

Interactive image processing and pattern recognition of digitized flow patterns /

Pieter J. M. Kerstens
Lehigh University

INTERACTIVE IMAGE PROCESSING AND PATTERN RECOGNITION OF DIGITIZED FLOW PATTERNS

by

Pieter J. M. Kerstens

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree

Master of Science

in

Manufacturing Systems Engineering

Lehigh University

1986

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Manufacturing Systems Engineering.

12/08/85
(date)

Professor in charge

Director of MSE Program

Chairman of CSEE Department

ACKNOWLEDGEMENTS

I would like to thank Professor Donald Rockwell of the Mechanical Engineering Department for his support and guidance, and for carefully reviewing this thesis report. I would also like to thank JoAnn Casciano for patiently typing this report and Carlos Gomez, Professor Roger Nagel, and the staff and secretaries of the MSE Program for their kindness, advice, and help during this year. In addition, I thank Philips Laboratories for their support and for letting me join this worthwhile program. Last, but not least, I would like to thank my wife, Anja, for her patience and warm support, without which this work would not have been possible.

TABLE OF CONTENTS

Table of Contents (Continued)

Table of Contents (Continued)

GLOSSARY OF NOTATION

Symbol	Description
A	Interval limit for length of curve
A_k	Interval limit for number of samples in curve
$A^*(k)$	Element similarity
$A(k,\tau)$	Element similarity for shifted curves
c_n	Fourier coefficient
$D_{x,old}$	Absolute displacement of original curve in x direction
$D_{x,new}$	Absolute displacement of smoothed curve in x direction
$D_{y,old}$	Absolute displacement of original curve in y direction
$D_{y,new}$	Absolute displacement of smoothed curve in y direction
$f(\ell)$	Continuous filter function
$f(k)$	discrete filter function
L	Curve length
L_1	Curve length of shortest curve
L_2	Curve length of longer curve
ℓ	Length parameter
N	Number of elements in curve
R	Interval limit for shift parameter
$R_{\phi_1\phi_2}(\tau)$	Cross-correlation function
$R^*_{\phi_1\phi_2}(\tau)$	Normalized cross-correlation function
$S(\ell)$	Smoothed curve
$S^*_{\phi_1\phi_2}$	Similarity function

Glossary of Notation (Continued)

Symbol	Description
$S^{1}_{\phi_1\phi_2}(\tau)$	Similarity function for shifted curves
$S_{\phi_1\phi_2}(\tau)$	Modified similarity function for shifted curves
$x(\ell)$	Continuous x coordinate function
$x_s(\ell)$	Smoothed continuous x coordinate function
$x_{i,ave}$	x coordinate of point i of average curve
x_{ij}	x coordinate of point i of curve j
$\Delta x(k)$	Relative discrete displacement in x direction
$\Delta x^{*}_{s}(k)$	Smoothed $\Delta x(k)$
$\Delta x_s(k)$	Modified smoothed $\Delta x(k)$
$y(\ell)$	Continuous y coordinate function
$y_s(\ell)$	Smoothed continuous y coordinate function
$y_{i,ave}$	y coordinate of point i of average curve
y_{ij}	y coordinate of point i of curve j
$\Delta y(k)$	Relative discrete displacement in y direction
$\Delta y^{*}_{s}(k)$	Smoothed $\Delta y(k)$
$\Delta y_s(k)$	Modified smoothed $\Delta y(k)$
W	Interval limit for length of filter
W_k	Interval limit for number of samples in filter
w_j	Weighting factor for curve j
$\alpha(\ell)$	Direction function for larger semicircle
$\beta(\ell)$	Direction function for smaller semicircle
γ_n	Angle with unit of radians

Glossary of Notation (continued)

Symbol	Description
λ	Integration variable
μ	Integration variable
τ	Shift variable
$\phi(\ell)$	Continuous tangent angle function
$\phi^*(\ell)$	Continuous function, periodic with period L
$\phi(k)$	Discrete tangent angle function
$\|\|\phi\|\|$	Norm of ϕ

ABSTRACT

In this thesis report, interactive algorithms to aid in the analysis of fluid flows are presented. Special functions and algorithms to average, smooth, and calculate the similarity between digitized curves were developed. The developed routines process the images in the spatial domain, thereby eliminating the need to calculate discrete Fourier and inverse Fourier transforms. The algorithms are effective, efficient, and fast. As an integral part of the algorithms, special data buffer routines for the effective data manipulation of curves, as well as cursor routines, were developed. An arbitrary set of frames consisting of curves, or an arbitrary set of curves, can be averaged or smoothed. Curves can be smoothed with a modified, variable convolution filter. A special function makes it possible to express the similarity of two curves in a numerical value. This technique can be used to study time effects in fluid flows. Noise reduction can be obtained by averaging and smoothing a set of curves.

1. INTRODUCTION

This report describes the underlying theories and algorithms for an interactive image processing and pattern recognition program that can be used in the analysis of digitized flow patterns. To aid in fluid flow analysis, a television/computer system is set up in the fluid mechanics laboratory of the Mechanical Engineering Department of Lehigh University. The actual setup is described and explained in Chapter 2.

Noise and disturbance reduction through successive averaging of images is one of the main goals of the program. For this purpose the digitized images are stored in a mass storage device and are processed off-line. In the future, it should be possible to extract features from the flow patterns that can be used in the generation of a data base. The setup will facilitate the analysis process by increasing speed and accuracy and by providing the possibility of new analysis techniques.

Currently the hardware and software to generate digitized curves from the actual flow patterns is in place. Chapter 2 describes the flow visualization technique that is used in conjunction with a television system and synchronized lighting. The television images are digitized, then preprocessed, and algorithms employed to generate the digitized curves in binary form. Once the curves are in this form, it is possible to carry out a number of operations on them.

The objectives of the thesis work presented in this report are to:

a) develop theory and algorithms to average a set of curves.

b) develop theory and algorithms which make it possible to filter/smooth a curve.

c) develop theory and algorithms that are capable of expressing the similarity between two curves in a numerical value.

d) combine the above in a flexible and interactive software program that can aid in the analysis of flow patterns.

A selection from among the presently available analysis and processing techniques has to be made in order to find the most suitable one. Additionally, it is necessary to develop new principles for those functions that cannot be performed by currently known techniques. Capacity, in terms of the number of frames and curves that should be processed, as well as efficiency and accuracy requirements, are important considerations during the selection of the methods.

Based upon previous work done at Lehigh University, and the scope of this thesis, several assumptions were made. The first assumption is that the digitized curves are available. This means that there is a complete set of frames available, each containing well defined digitized curves. Secondly, it is assumed that these curves are stored in a chaincode representation (see Chapter 3). Finally, a highly interactive program is required for further

analytical work. Chapter 2 describes the system setup, the presently known analytical techniques, the more suitable methods for fluid flow analysis. Chapter 3 deals with the data representation and interactive routines that were developed to facilitate the processing, filtering, and recognition of digitized curves in an efficient manner. Chapters 4 and 5 deal with the developed processing and recognition techniques. Chapter 4 discusses the averaging routines actually used, while Chapter 5 presents the theories and algorithms that were developed so the digitized curves can be filtered/smoothed and recognized. Future work and conclusions are discussed in Chapter 6. The actual program codes can be found in Kerstens (1985).

2. IMAGE PROCESSING

2.1 System Setup

2.1.1 Flow visualization

The fluid mechanics laboratory of the Mechanical Engineering Department at Lehigh University is equipped with the setup of Figure 2.1. The experimental setup consists of a channel through which a fluid (water) flows from left to right. A conducting wire is inserted at the beginning of the channel, and a pulsed high voltage is connected to the wire. This causes electrolysis of the water passing over the wire and generates hydrogen bubbles. The hydrogen bubbles propagate with the fluid and form the timelines.

The triangular obstruction in the middle of Figure 2.1 is a stationary or oscillating body that distorts the fluid flow and thus the timelines. The timelines contain information about the fluid flow (velocity, both amplitude and direction) and can be used to characterize and analyze the flow. Light is reflected at the timelines and regions of high intensity are created at their position. A video camera is placed above the channel and records the illuminated timelines. The times at which pictures are taken, can be triggered in such a way that they are synchronized with a particular position of the oscillating body. After recording on the television, the image of timelines is digitized by an analogue/digital convertor (ADC) with 256 quantization levels (8 bits). The resolution per

frame is 211 x 165 pixels. A frame grabber grabs the frames which are then stored on tape.

2.1.2 Preprocessing

A good introduction and explanation of several important aspects of image processing can be found in Pavlidis (1982). The digitized image must be processed before the actual curves can be retrieved (see Figure 2.1). Several methods are currently in use. In the first method (Gumas 1985) the signal is filtered and then operated upon by a threshold operation, which results in the generation of a binary image. The resulting timelines are, however, still relatively wide. The thickness can be reduced by thinning algorithms (Pavlidis 1982 and Gumas 1985). The resulting pictures have a quantization of 1 bit (2 levels) and contain the timelines. In the second method (Gumas 1985) a curve tracking algorithm processes the original frames with 8 bit quantization levels and "tracks" the timelines. The output of this algorithm is a set of curves (timelines) in chain code form (see Chapter 3). An interesting method relevant to the foregoing is presented in Rao (1980). Fingerprint patterns are very similar to flow patterns and a complete system for the classification of fingerprints is presented in Lerner (1983).

2.1.3 Available hardware and software

The preprocessed images are stored in computer files on the

VAX 11/780 computer of the Mechanical Engineering Department of Lehigh University. Several high resolution color terminals (VS11) are connected to this computer and are used in the advanced processing of the flow patterns. The resolution of the terminals is 512 x 480 pixels and they are capable of displaying 16 colors (of which one is the background color). Some graphics routines, for displaying basic entities (lines, circles, splines, points, windows, labels, etc.) are available and are well explained in Ozsoy (lab 1983).

2.1.4 Developed graphics software

To enhance the interactiveness of the developed program, and to increase its flexibility, it was necessary to develop some additional graphics routines. A set of windows was created (Kerstens 1985) to display the curves and related information, as well as to display system messages. Special buffer routines, that can display and remove arbitrary curves in an efficient manner, had to be implemented. Another requirement was a special cursor routine for picking curves. These routines are explained in Chapter 3.

2.2 Analysis Techniques

There are at least three techniques available that could be used in the analysis of flow patterns: Fourier descriptors, classification techniques, and spatial averaging. The remainder of this section will briefly discuss each of these techniques.

2.2.1 Fourier descriptors

Fourier descriptors are often used in pattern recognition applications because special, modified techniques using Fourier descriptors make the descriptors independent of orientation, position, or scale of the object to be described. They can be made solely a function of the shape of the object (Granlund 1972 and Zahn 1972). To understand how Fourier descriptors can be used in the analysis of flow patterns, consider Figure 2.2. The shape of the shown curve is completely described by its "tangent angle function":

$$\phi(\ell) \qquad \ell \varepsilon [o,L] \tag{2.1}$$

where

$\phi(\ell)$ is the tangent angle of the uniform continuous curve at point $\ell \varepsilon [o,L]$.

ℓ is the length measured along the curve from its starting point to the point of interest.

L is total length of the curve measured along the curve.

The tangent angle function together with the start coordinates of the curve uniquely specify the curve. Defining a function ϕ^* such that:

$$\phi^*(\ell + nL) = \phi(\ell) - \frac{\ell}{L}\{\phi(L) - \phi(o)\} \text{ for } \begin{cases} \ell \varepsilon [o,L] \\ n = 0, \pm 1, \pm 2, \ldots \end{cases} \tag{2.2}$$

The function ϕ^* is periodic and can be expanded in a complex Fourier series:

$$\phi^*(\ell) = \sum_{-\infty}^{\infty} c_n \exp(jn\gamma_n) \tag{2.3}$$

where,

$$\gamma_n = \frac{2\pi\ell}{L} \tag{2.4}$$

and,

$$c_n = \frac{1}{L} \int_0^L \phi^*(\ell) \exp(-jn\gamma_n) d\ell \tag{2.5}$$

The set of c_n's and γ_n's are called the Fourier descriptors: $\{c_n, \gamma_n\}_{n=-\infty}^{\infty}$. In reality, we have to work with a truncated (finite) set of Fourier descriptors:

$$\{c_n, \gamma_n\}_{n=-N}^{N} \tag{2.6}$$

When this truncated set is used in the reconstruction of the original curve, the result will be an approximation, and the difference depends on the number of coefficients that is included in the set.

Averaging of curves can be performed by averaging the Fourier descriptors. The curves can be filtered by truncating the set of Fourier descriptors and/or by multiplying them with a weighting function (Gumas 1985). Similarity coefficients can be derived if the Fourier descriptors are made independent of the orientation and position of the curve.

2.2.2 Classification techniques

The following discussion is mainly based upon the works of Rao (1980) and Lerner (1983). Fluid flows and fingerprints seem to have a lot in common in terms of their image structure. Both

fluid flow patterns and fingerprint patterns consist of a set of curves or ridges that are to some extent parallel in nature. Imperfections can create gaps in the curves of both patterns and it may be difficult to track the curves due to some ambiguity in direction (Gumas, 1985; Rao, 1980).

In fingerprint classification systems, a preprocessor first averages over small areas of the image to find a mean level and then a threshold operation divides the points into two levels: black and white. After completion of this operation, directional operators determine the direction of the ridges, and ridge points are linked up to eliminate minor breaks caused by poor inking or skin pores. In the next step features are extracted. These features mark the endpoints and the points where the lines that form a ridge make an angle with each other (see Figure 2.3). Analysis of the extracted features makes it possible to classify the prints.

The preprocessing techniques used in fingerprint analysis seem to be directly applicable to fluid flow analysis. However, the classification techniques that are used are a syntactic approach to the identification problem. Therefore, this technique might be useful in identifying certain fluid flows, but it cannot be used for detailed analysis of the fluid flows. It is also impossible to average curves or to calculate similarity coefficients with this technique. The preprocessor used in this technique provides some means of smoothing or filtering the curves by averaging the direction of the curves over small areas of the image.

2.2.3 Spatial Averaging

In spatial averaging, the curves are averaged directly in the spatial domain. By taking specific points on two curves (see Chapters 3 and 4) and taking the averages of corresponding points on both curves, the average curve is calculated. The data representation that is used in the averaging process can also be used in the developed smoothing and similarity calculation algorithms (see Chapter 5). The technique is very accurate, relatively fast, and can be applied to an arbitrary number of curves.

2.3 Method Selection

The Fourier descriptor method requires the calculations of discrete Fourier transforms and discrete inverse Fourier transforms. These computations are relatively time consuming and costly. Furthermore, smoothing or filtering of the curves by truncating the set of Fourier descriptors will not preserve the endpoints of the curves (Gumas, 1985). Data storage requirements can be reduced by storing only a few of the Fourier descriptors per curve where one has to make a trade-off between the number of descriptors to be stored and the accuracy of the representation. However, these savings might not be as large as originally envisioned. Consider the following example. Suppose one gets a satisfactory representation by only storing 10% of the Fourier coefficients. Since every coefficient contains both amplitude and phase information, it is

represented by two real numbers (two Fourier descriptors out of the set defined in paragraph 2.2.1). In standard precision FORTRAN 77, each real number requires four bytes of memory. So if N is the number of coefficients, then one needs 0.10 x N x 2 x 4 = 0.80 N bytes of memory to store 10% of the coefficients. Representing the curve by a chaincode (see Chapter 3) is possible with one byte of memory per element, because each chaincode element is an integer number between one and eight (see Chapter 3) and thus the total amount of memory needed is N bytes. Similarity calculations are not easily implemented with Fourier descriptors, since in principle these coefficients are dependent on such factors as scaling, rotation, and translation. However, the special function ϕ^*, that was defined in paragraph 2.2.1, is independent of the orientation of the curve and is a principal candidate function for use in similarity measurements.

Classification techniques are not very accurate and cannot be used for averaging curves. This method also cannot be used to calculate the similarity between two curves. Spatial averaging on the other hand seems to overcome all the shortcomings of the other methods. It is very accurate and fast. The data representation that is used in this method is very efficient and preserves the original curves, including the start and endpoints. The developed smoothing algorithm proves that it is possible to smooth/filter curves in the spatial domain while preserving the start and endpoints

(see Chapter 5). Finally, similarity calculations can also be performed directly in the spatial domain (see Chapter 5) and this method is, therefore, the most suitable of the three techniques for the analysis of fluid flows.

3. DATA REPRESENTATION AND INTERACTIVE ROUTINES

In this chapter several aspects of the curve data representation are explained. Some special routines were developed for the efficient storage of curves, and for the interactive selection of the curves from a terminal screen. All these routines are presented in this chapter as well.

3.1 Curve parameterization

3.1.1 Chain codes

Every curve consists of a set of pixels. They lie next to each other and together form one particular curve. One way to represent each curve could be a recording of the x and y coordinates of the center of each pixel. The disadvantage of this technique is that we have to store two entities per pixel (see Figure 3.1). Another technique uses chain codes (Pavlidis 1982). Figure 3.2 shows how the chain code elements are derived. Each pixel in the curve has at least one neighbor. The neighbor pixel can be any one of the eight neighbor pixels shown in Figure 3.2. By assigning a different number to each of the eight pixels, one can uniquely identify where the neighboring pixel is positioned. Each curve is tracked from its start point to its end point, and the position of each pixel is recorded in the form of a chain code that is derived from the position of the pixel with respect to its preceding pixel. The only exception is the starting pixel. Since this pixel does not have a

predecessor, its x and y coordinates are recorded. Figure 3.3 shows a curve and the so derived chain code representation.

It is obvious that the chain code representation is a discrete representation. However, no information is lost since the original curve was already digitized by the video processing equipment. Since only one element per pixel is stored (with exception of the starting point), the data or curve representation is very efficient. A disadvantage is that one cannot directly relate a pixel's chain code value to a particular position on the screen. One always has to track the curve from its starting point to find out where a particular pixel is located.

3.1.2 Element length

Each curve consists of a number of elements, where an element is defined as that part of the curve represented by the connecting line between the centers of two succeeding pixels. The length of each element depends on the configuration of the two pixels. Considering Figure 3.4, one can see that the distance between the two centers of two succeeding pixels is equal to 1 (after normalization) if they have one side in common and equal to $\sqrt{2}$ if they have only one common corner. By adding the length of all elements that form a curve, one can calculate the total length of the curve.

3.1.3 Program input

The starting point and chain code representation of each curve

are inputs to the program. The end of a chain code representation is identified by "0". For each curve the program generates three arrays. The first two arrays contain the x and y coordinates of each pixel. The third array stores the length, measured along the curve, of that part of the curve that is located in between the pixel of interest and the curve's starting point. If one uses array L for this purpose (see Figure 3.5), then the value of L(1) is either 1 or $\sqrt{2}$ depending on how the second pixel is located with respect to the first. The value of L(2) is either 2, $1 + \sqrt{2}$, or $2\sqrt{2}$ depending on the location of the second pixel on the curve with respect to the first pixel and the location of the third pixel with respect to the second one. There is another possibility: L(2) can also equal 0 if the curve consists of only two pixels.

The so formed arrays are used in the averaging, smoothing and correlation routines (see Chapters 4 and 5).

However, a different notation is sometimes more useful. In this notation the chain code values are converted into two arrays containing the Δx and Δy values of all the chain codes that form a particular curve. This is shown in Figure 3.6. Each chain code element simultaneously represents a Δx and Δy value that can be equal to either -1, 0, or +1. A chain code value of 9 is included to represent those cases where the next pixel is actually on top of its predecessor. This can occur during the smoothing and averaging operations (see Chapters 4 and 5). Since these points do not

contain useful information, they are usually removed from the chain code representation. Figure 3.7 shows how a set consisting of Δx and Δy values can be converted to a chain code description. Because FORTRAN does not accept negative or zero subscript values, the value 2 is added to the Δx and Δy values and the corresponding chain code value is found by looking in an array containing these values. The first subscript of this array is the $\Delta x + 2$ value and the second subscript is the $\Delta y + 2$ value.

The $(\Delta x, \Delta y)$ representation is especially useful in the available display routines, because they require the incremental x and y values as input variables (Ozsoy 1983).

3.1.4 Freeman's corner cutter matrix

By using Freeman's corner cutter matrix (Freeman, 1961; Gumas, 1985), one can remove the jaggedness of a curve. Figure 3.8 shows that certain pixels can be removed (the black pixels in this figure) without significant loss of information. In fact the curves look much smoother after this routine is applied (see Figure 3.8). After applying this routine, the curves never change direction over more than $\pm 45^{o}$ going from the preceeding two pixels (giving the current direction) to the next pixel in the curve. The method repeatedly replaces two adjacent chain code elements by the new element(s) found in Freeman's matrix (see Figure 3.9). The second (or only) new element is used in the next replacement step. In Figure 3.10 the routine is applied to the curve of Figure 3.8. Sometimes a

single iteration is not enough and the routine has to be repeated until the chain code no longer changes. Although the routine smooths the curves, it only does so on a local basis. Freeman (1961) also shows how the routine can be used to find the shortest path between the curve's startpoint and endpoint. This method provides maximum smoothing of the curve. However, unlike the algorithm presented in Chapter 5, no means of arbitrary smoothing is provided by Freeman's matrix.

3.2 Display routines

Although some basic graphic routines are already available (Ozsoy 1983), some twenty-five additional graphic routines were developed to aid in the fluid flow analysis process (Kerstens 1985). These routines create a very interactive environment, set up and clear the screen, allow the user to change the center of display and scaling factors, provide many options, and quickly allow the user to display/average/smooth/correlate (an) arbitrary frame(s)/ curve(s) etc. Most of these routines are explained in Kerstens (1985).

However, in order to keep track of which curves are displayed where on the screen, in an efficient manner, and to enable the implementation of the averaging, smoothing, and correlation routines, special buffer and cursor routines were developed. Since these routines are fairly complicated and represent a major part of the total FORTRAN code, the developed algorithms are presented and

explained in the remaining two paragraphs of this chapter.

3.3 Curve storage buffer

After a curve is displayed on the screen, it is often used in some further processing. Suppose the displayed curve is the output of previous processing routines; then it is necessary to store the relevant data of this curve so one can use it in the future without having to recalculate it. To accommodate this, special buffer routines that store all relevant data of the curves displayed on the screen were developed. Figure 3.11 shows the setup of the buffer while its components are explained in the remainder of this paragraph.

3.3.1 Buffer pointers

The stack contains the main curve data. It contains the Δx and Δy arrays, the curve's startpoint, the curve's color, and a lot of other information for each curve that is displayed on the screen (see Figure 3.11). Since curves are continuously removed and added to the list of displayed curves, the stack continuously changes. So the stack has to be updated quite frequently, and one has to keep track of where a particular curve is stored. Suppose the current number of displayed curves is N. One way of storing the curves would be to store them in the first N positions of the buffer (see stack in Figure 3.11), in the order they appeared on the screen, and to remember the current number of displayed curves N.

Assume, however, that one wants to remove one of the first curves in the stack from the screen. Since this curve is no longer shown on the screen, it also has to be removed from the stack (otherwise a stack overflow would quickly occur). This in turn creates a gap in the stack that has to be closed. Suppose the third curve stored in the stack is removed from the screen. To close the created gap, the N-3 curves that were placed on the screen, after the third curve was put on the screen, have to be shifted down one position in the stack because the third curve was removed. Since each curve can contain over 200 data points, this algorithm would be very inefficient.

An alternative would be to remember where each curve is stored in the stack, remember where there are gaps in the stack, and to fill up the gaps with data of the new curves being displayed on the screen. Basically the new curve data would overwrite the old curve data. This is a very efficient algorithm in which no data is re-positioned in the stack. Implementation of this algorithm requires that one keeps track of where the curve data is positioned. To understand how this can be done efficiently, one first has to consider how the buffer is going to be used.

As mentioned in Chapter 2, the data is generated in the form of frames each consisting of several curves. Therefore, it is more than likely that at some time one wants to display simultaneously a complete frame, or a complete set of curves for that matter. If

this is the case, a sequence of curves should be stored in the buffer. Keeping track of the sequences would make it possible to delete any particular sequence from the buffer (and thus the screen). So every sequence of curves is stored and removed in one shot. However, if this is the case, the only thing to remember is the stack addresses in which the curves of particular sequence are stored and the total number of sequences that are currently displayed on the screen. Figure 3.11 shows how this is implemented. The stack addresses of the stored curves is stored in a one-dimensional array in the same order as they appear on the screen. This is the array labeled "curve-addresses" in Figure 3.11. Every curve of every displayed sequence is stored in this array. To keep track of where a particular sequence starts in the "curve-addresses" array, a second one-dimensional array, containing the sequence positions in the "address-array", was created. This array is also shown in Figure 3.11 and is labeled "sequence-position". Stored in this array is the address of the last curve out of each sequence in the "curve-addresses" array.

Suppose the address of the last curve of sequence N-1 is stored in position x of the "curve-address" array and the address of the last curve of sequence N is stored in position Y of the "curve-address" array. Then the total number of curves in sequence N is Y - X. So by keeping track of the last address of each sequence in the "address-array", it is possible to locate the stack-addresses

of all the curves in a particular sequence. The only exception is the first sequence. One cannot find the number of curves in this sequence by substracting the position of the last curve in the previous sequence from the position of the last curve in this sequence, since this sequence does not have a predecessor. However, for this sequence the number of curves is equal to the pointer value and one can use this value instead. The pointer value of the current sequence is updated each time one adds a curve to this sequence. After completion, the sequence is displayed on the screen (note: a sequence was defined to be a set of curves that are transferred to the screen simultaneously). This results in a new sequence being started and this in turn means that the pointer in the "sequence-array" that was identifying the last curve of the then current sequence, is now pointing to the last curve of the just completed sequence. A "sequence-counter" is keeping track of the number of sequences and is pointing to the position of the sequence-position pointer of the current sequence in the "sequence-position" array (see Figure 3.11). As mentioned before, to keep track of the curve's stack-addresses is not enough. It is also important to keep track of the gaps in the buffer, because new curves have to be stored in these positions of the stack. To do so efficiently, the available stack-addresses are stored in a one-dimensional array called "available-addresses" (see Figure 3.11). A pointer keeps track of how many of the addresses are still available. Each time a new

curve is stored, an address is removed from the top of the "available-addresses" array, the pointer value (pointing to this array) is lowered by one, thereby pointing to the next available stack-address, the sequence-position pointer is incremented by one, thereby pointing to the next position in the "curve-addresses" array, and the address that was just removed from the "available-addresses" array is stored in the "curve-addresses" array, thereby pointing to the gap in the stack where the data of the new curve will be stored.

So, a minimum of data manipulation takes place each time a new curve is stored. All the relevant curve data are now stored in the stack.

If a curve is removed from the stack, a similar routine takes place. One can remove any sequence from the screen and thus the stack. If a sequence is removed, the stack-addresses of all the curves in that sequence are removed from the "curve-addresses" array and stored in the "available-addresses" array. The pointer to the "available-addresses" array is updated and is thus still pointing to the first available-address in that array. The gap in the "curve-addresses" array is removed by shifting up all the curve-addresses of the curves in the sequences that followed the removed sequence. Accordingly, the sequence-position pointers to these sequences are adjusted. The gap in the "sequence-position" array (created by the removal of the pointer to the removed sequence) is removed in a

similar fashion. Since one sequence was removed, the "sequence-counter" value is decremented by 1. The data of the curves that were removed is not removed from the stack. If a new curve is stored at this position in the stack, its data will simply be written over the old curve data. Again a minimum amount of data manipulation was required to remove a complete sequence of curves from the stack and thus the screen.

The buffer routines form an integral part of the image processing algorithms. They provide a means of efficient curve manipulation and data storage. Without them the flexibility of the routines would be greatly reduced and the implementation of similarity measurements (see Chapter 5) would become a very difficult task.

3.3.2 Stored curve data

The stack consists of three arrays (see Figure 3.11). When a curve is stored, the data of the curve is divided over the three arrays. The stack-address of the curve is the same for all three arrays. The first array stores the Δx and Δy values (see section 3.1.3) of the curves. This is a three-dimensional integer array. The three subscripts of this array are: curve-address (first subscript), Δx or Δy (second subscript is 1 for Δx and 2 for Δy), and pixel number (third subscript). The second array is a two-dimensional integer array that stores other relevant data of a curve. The two subscripts of this array are: curve-address (first

subscript), and relevant data number (second subscript). The relevant data is always stored in the same order (see Figure 3.11). The first data element is reserved for the x-coordinate of the startpoint of the curve. The second data element stores the y-coordinate of the startpoint of the curve. Also stored are the current color of the curve, the number of Δx elements (or Δy elements) in the curve (this number is equal to the number of pixels in the curve -1), the label-type, label-switch, curve number, frame number, and window of display. A curve can be identified by looking at its curve and frame number. Each frame has a unique number and each curve within a particular frame also has a unique number. The curve numbers are displayed either above or below the starting points of the curves. Studying Figure 2.1 reveals why this is convenient. The general direction of the curves in the upper half of the screen is always downwards while the general direction of the curves in the bottom half of the screen is always upwards. So curve labels can always be put above the startpoint of the curve if the curve is in the upper half of the screen and below the startpoint if the curve is located in the bottom half of the screen. The curve labels consist of the curve's number and an optional character telling if the curve is the output of an averaging (see Chapter 4) or smoothing (see Chapter 5) operation. This information results in a unique label code that is stored in the label-type location of the second stack array. A label can be switched on or off and its current

status is stored in the label-switch location of the second stack array. The display-window location of the stack array keeps track of the window in which the curve is displayed (either full screen or not, see Kerstens 1985).

The third stack array is a two-dimensional real array. It contains the scaling factor and center of display (x and y coordinate) for each curve. These values can be changed by the user which enables him to zoom-in on particular portions of a curve or to reposition the curves on the screen. Rather than recalculating all the new Δx, Δy, and startpoint coordinate values, these display parameters are stored instead. Again, the first subscript is the curve address, while the second subscript is pointing to the data location of interest.

3.3.3 Buffer routines

There are four buffer routines: BUFINIT, BUFSEND, BUFDISP, and BUFERASE. The first routine, BUFINIT, initializes the buffer. It resets all the pointers and fills up the "available-addresses" array with all the stack-addresses. This routine is called in the beginning of the main program and in the clear-screen routine. The second routine, BUFSEND, stores all the relevant data of a curve in the buffer. It generates warning messages when the number of sequences reaches its maximum (currently 10) or when the number of curves reaches its maximum (currently 30). If the capacity of the buffer is exceeded, no curve data is stored, no additional curves

are displayed, and the user is notified of the fact that, either the stack is full or the number of sequences has reached its maximum, and that no data was stored. The dimensions of the appropriate arrays can be increased, if necessary, to facilitate the user's storage requirements.

The third routine, BUFDISP, displays the curves in the current sequence and updates the "sequence-counter".

The last routine, BUFERASE, will erase a specified sequence (set of curves) after some specified delay. It also has a refresh option. Curves are removed from the screen by redisplaying them in the background color. Therefore, if one of the removed curves crosses a displayed curve, the latter will have a gap after the removal process is completed. By refreshing the screen (redrawing the still displayed curves in their own color) after the removal operations, these gaps disappear. Finally, during the refresh operation, the labels can be either removed or displayed.

3.3.4 Display_map

An exact copy of the displayed curves on the screen is maintained in a two dimensional byte array. The resolution of this "display-map" array is the same as that of the screen: 512 x 480 locations. Each time a pixel of a curve is displayed on (removed from) the screen, the stack-address of that curve is stored in (removed from) the corresponding location of the "display-map" array. By just storing the stack-address of the curve, it is possible to keep the

size of the "display-map" array down to a relatively modest value, while one can still find all the relevant curve data by looking at the stack locations that correspond with this stack-address. These operations take place in the BUFDISP and BUFERASE routines. The "display-map" is used in the developed cursor routines which are discussed in the next paragraph.

3.4 Cursor routines

The similarity routines require that the curves, between which a similarity value is calculated, are picked from the screen by the user. The reason for this is simple: to give the user the flexibility of calculating the similarity between arbitrary curves, originals, averaged, smoothed, or a mix of them. A different type of implementation would probably be a burden to the user and would not give him the flexibility of a cursor routine. However, such a routing was not available and had to be developed. A routine that returns the x and y coordinates of the picked pixel is available (Ozsoy 1983) and is used in the curve picking routine.

3.4.1 Search routine

The search routine must be fast and efficient, have a high resolution (meaning it must be able to differentiate between two curves that are close to each other), and find the curve that is closest to the cursor. To find a curve, the cursor is shown on the screen and the x and y coordinates of the pixel picked by the user are returned to the search routine. The search routine uses

these values to look in the display map (see section 3.3.4) to see if this pixel belongs to a curve. This is the black center pixel of Figure 3.12. If it finds a stack-address (a number ≠ 0) in the display map the curve is found, and its stack-address is used to find the required curve data in the stack. If no stack-address is found, the search routine will search for the nearest pixel that is part of a curve, its equivalent; moreover, it will look for the nearest location in the display map array containing a stack-address. Figure 3.12 shows the order in which the locations are checked. The locations with the number 1 in them are the ones closest to the center location and are checked first. They are checked in a counterclockwise fashion starting with the location in the second quadrant. If no stack-address is found, the locations numbered two and three are checked in a similar fashion (starting with two). If still no address is found, the locations numbered four are checked. Figure 3.12 shows that each quadrant contains two locations that are numbered four (both locations are equally far from the center location). Again these locations are checked in a counterclockwise fashion. For each location in the second quadrant with a number between one and forty-one (see Figure 3.12), its relative coordinates with respect to the center location are stored in a look up table. Figure 3.13 shows part of this table. The relative coordinates of the first four numbers (five locations) in each quadrant are shown in this table. From the table one can see that the relative coordinate

values of the locations in the third quadrant can be obtained from the relative coordinate values of the equivalent location in the second quadrant. The required transformation routine is: invert the sign of the relative y coordinate and flip this value with the relative x coordinate. The relative coordinate values of the locations in the fourth quadrant can also be obtained from the values in the second quadrant. The transformation required here is: invert the sign of both the relative x and y coordinates. Finally, to obtain the relative x and y coordinates of the locations in the first quadrant from the values in the second quadrant, one has to invert the sign of the relative x coordinate and flip this value with the relative y coordinate.

So only the relative coordinates of the second quadrant have to be stored in the look up table since the other values can be obtained with a simple transformation. The first forty-one numbers in the second quadrant represent seventy-five locations (see Figure 3.12). So by looking at the seventy-five locations in the second quadrant and the two hundred twenty-five locations in the other three quadrants, the search routine looks at three hundred locations (plus the center location) to find a curve. The total screen consists of 512 x 480 = 245,760 pixels so the search routine is checking approximately 1 out of 800 pixels. For a thirteen inch monitor with an aspect ratio of 3:4, the total viewing area is 7.8 x 10.4 = 81.12 square inches. So the total area checked by

the routine is:

$$\frac{301}{245,760} \times 81.12 \approx 0.1 \text{ square inches} \qquad (3.1)$$

Since the checked area approximates the shape of a circle, the diameter of this area is approximately:

$$2x\sqrt{\frac{0.1}{\pi}} = 0.36 \text{ inches} \qquad (3.2)$$

This provides more than enough resolution and will make it easy for the user to pick the desired curve even if the curves are very close to each other.

The routine will find the curve that is closest to the center of the cursor. However, if a curve is further than approximately .18 inches from the center of the cursor, the routine will not find the curve. To compensate for this, the search is expanded to check the locations on the horizontal, vertical, and two diagonals going through the center of the cursor (see Figure 3.14). Again, the routine checks the locations in a counterclockwise fashion. After completion of the local search, it starts with the relative coordinates (0, 10) above the cursor, then checks the location (-10, 10) on the diagonal, then the location (-10, 0) on the horizontal, etc. Although the routine skips a lot of locations (to keep it efficient), it generates a "star" of checked locations and should almost always find a curve. If still no curve is found, the user has to try again. In the worst case when no curve is found, the cursor is positioned in the center of display, the

additional number of checked positions is equal to 1872 and the total number of checked locations is 1872 + 301 = 2173. So approximately 1 out of 113 pixels is checked in the worst case situation (less than 1%) keeping the routine still very efficient.

3.4.2 Display_update

As soon as a curve is found by the search routine, the routine is terminated and the curve's stack-address is returned. Because the curve's stack-address is known, all relevant data can be found in the stack. This data is used to redisplay the curve in a different color immediately after it is found. The user can accept or reject the curve. If the curve is rejected, it is again redisplayed in its old color. If the curve is accepted, its data is used in the operation described in the following.

4. CURVE PROCESSING

One of the main objectives of this thesis was to develop an algorithm capable of averaging a set of curves. The developed algorithm and the required support routines are presented in this chapter.

4.1 Length calculations

Before any averages can be calculated, some consideration has to be given to the relevant length of the curves. It is also important to find the right points on each curve that must be used in the averaging operation. In the remainder of this paragraph these topics are discussed.

4.1.1 Normalized versus shortest curve lengths

Two curves almost never have the same length (see Figure 2.1). Therefore, before two or more curves are averaged, some thought has to be given to how the difference in length is to be handled. One can either normalize the curve's lengths or one can average the curves over the length of the shortest curve.

If two or more curves are averaged, a number of points on one curve are compared with the same number of points on the other curves. If the length of the curves is normalized, these points are spread out over the entire curve. The length of a section between two of the points (measured along the curve) is the same for all sections of one curve. So if one curve is longer than

another curve, these sections are also longer. A justification for using normalized lengths could be the assumption that the curves started out to have the same length, but the disturbance in the flow stretches some curves more than others. By normalizing the curve's length (spreading out the points over the entire length of the curve) one can compensate for this effect. This is shown in Figure 4.1. Another possibility is to assume that the length of a section between two points does not change from curve to curve. So a longer curve just has more points. Averaging requires that the same number of points on each curve is used. So if the shortest curve has N points, only the first N points of the other curves are used. Since the other points are not available for the shortest curve, one cannot average the remaining points on the longer curves. This is shown in Figure 4.2.

4.1.2 Length data

To find the location of the points on each curve (see paragraph 4.2), it is important to know for each pixel what the length of the curve section enclosed by this pixel and the startpoint of the curve is. Therefore, this value is calculated for each pixel in each curve and stored in a special array for each curve. If this value is known for pixel N, the value for pixel N + 1 is found by adding the distance between the centers of pixel N and pixel N + 1 (either 1 or $\sqrt{2}$, see sections 3.1.2 and 3.1.3) to the value of pixel N.

4.1.3 Resolution considerations

Since the same number of points of each curve is used in the average calculations, it is not immediately obvious how many points per curve have to be calculated. Too many points would make the algorithm inefficient, while resolution would suffer if the number of points is not high enough. Therefore, when normalized lengths are used, the number of points is made equal to the number of samples (pixels) in the curve with the highest number of samples. When lengths equal to the length of the shortest curve are used, the number of samples in the part of each curve that is being used in the averaging routine, is counted and the number of points is made equal to the highest number of samples in any one of those sections.

4.1.4 Binary search for sections of equal length

When the length of the shortest curve is used as a reference to determine what part of the longer curves is to be used in the averaging calculations, one has to calculate the number of relevant samples (pixels) in those curves. The length of, and the number of pixels in, the shortest curve are known. For each pixel, the distance (measured along the curve) to the curve's startpoint is known (see section 4.1.2). At some pixel, for every curve, this length will be approximately equal to the length of the shortest curve. The maximum error is $\frac{1}{2}\sqrt{2}$ which is half of the maximum distance between the centers of two pixels (see section 3.1.2).

This pixel is found by means of a binary search (very efficient). Suppose there are N + 1 pixels in the shortest curve. Starting at the startpoint of the curve, each pixel is assigned a number. The startpoint pixel is pixel 0 and the endpoint pixel is pixel N. Then the shortest curve consists of N elements where an element is the connecting line between the centers of two adjacent pixels. The length of each element is either 1 or $\sqrt{2}$ (see section 3.1.2). So the total length of the shortest curve is a value between N and $N\sqrt{2}$. If the shortest curve is a straight horizontal or vertical line, its length is N. A different curve having the same length has at least $N/\sqrt{2}$ pixels. (If this quantity is not an integer, the value is rounded to the nearest integer that is smaller than this value). The only time this minimum is reached is if the second curve makes an angle of 45^{o} (or 135^{o}) with the horizontal. This is shown in Figure 4.3.

On the other hand if the shortest curve makes an angle of 45^{o} with the horizontal, its length reaches the maximum value of $N\sqrt{2}$ (the shortest curve consisted of N elements). A different curve having the same length has at the most $N\sqrt{2}$ pixels (if this number is not an integer, it is rounded to the nearest integer that is larger than this value). The only time this maximum is reached is if the second curve is a horizontal or vertical line (see Figure 4.4). If the position or shape of the shortest curve is different from the ones described above, the minimum and maximum values will

not be reached. So if the number of elements in the shortest curve is N, the number of elements in the other curves being used in the averaging calculations is within the range:

$$\lfloor N/\sqrt{2} \rfloor \leq \text{elements used} \leq \lceil N\sqrt{2} \rceil \qquad (4.1)$$

Within this range there is a pixel whose distance to the curve's startpoint is equal to the length of the shortest curve L. This is shown in Figure 4.5. The exact location of this pixel is found with a binary search routine. Figure 4.6 shows this technique. The range of possible pixels is divided in two. The distance of the curve's starting point to this pixel is compared with the length of the shortest curve. If this length is less than the length of the shortest curve, this pixel becomes the new lower limit of the range of possible pixels. If the length was larger than the length of the shortest curve, this pixel becomes the new upper limit of the range of possible pixels. By repeating this technique, the right pixel will be found very quickly (see Figure 4.6), after which the number of pixels to be included in the averaging calculations is known.

4.2 Curve averaging

Averaging of curves is performed by averaging a large number of points on each curve. The number of points is equal to the maximum number of pixels in the curves (see section 4.1.3) and their location is found by the method that is presented in the next section.

4.2.1 Location of averaging points

The averaging routine requires a number of points that are spaced equally over the part of the curve that is to be included in the averaging operation. The spacing between the points is found by dividing the length of the section of the curve one is interested in (see section 4.1.1) by the total number of points minus one (see section 4.1.3). Since both the number of points per curve and the spacing between the points is now known, it is possible to calculate the absolute x and y coordinates of each point. Suppose there are N + 1 points numbered 0 through N. The distance (measured along the curve) between a certain point and the startpoint of the curve is equal to the points number multiplied with the interpoint spacing. Tracking the curve over this distance will give the point's position and thus coordinates. For each point and each pixel the distance to the startpoint is known. So it is possible to find the two pixels on each side of the point that are closest to that point. Since the x and y coordinates of each pixel are also known, it is now possible to calculate, by linear interpolation, both the location and coordinates of each point. This position usually does not coincide with any of the pixel centers. As an example consider Figure 4.7. The curve in this figure consists of seven pixels or six elements (only the centers of the pixels are shown). The length of the curve is $3 \times 1 + 3 \times \sqrt{2} = 3(1 + \sqrt{2})$. If seven points have to be equally spaced over this

curve, the spacing between them is equal to $3(1 + \sqrt{2})/(7-1)$ $\simeq 1.2$. The point locations are shown in Figure 4.7. Clearly the interpoint spacings are constant over the entire curve and only the first and last point coincide with a pixel center (this is always the case).

4.2.2 Calcualtion of the average curve

The average curve of a set of curves is found by averaging the equally spaced points one by one. To average a particular point on all these curves, the averages of both the x coordinates and the y coordinates of this point are calculated. Weighting factors can be assigned to each curve. By assigning equal weighting factors ($\neq 0$) to each curve, an unweighted average is obtained. Thus the coordinates of a point on the average curve are calculated by:

$$x_{i,ave} = \frac{\sum_{j=1}^{n} x_{i,j} w_j}{\sum_{j=1}^{n} w_j} \qquad (4.2)$$

$$y_{i,ave} = \frac{\sum_{j=1}^{n} y_{i,j} w_y}{\sum_{j=1}^{n} w_j} \qquad (4.3)$$

where

$x_{i,ave}$ = the x coordinate of point i of the average curve

$y_{i,ave}$ = the y coordinate of point i of the average curve

$x_{i,j}$ = the x coordinate of point i of curve j

$y_{i,j}$ = the y coordinate of point i of curve j

w_j = the weighting factor for curve j

n = the total number of curves that are averaged

This process is repeated for all the points of section 4.2.1 and for the startpoints of all the curves. The points on the average curve will not be equally spaced anymore. However, since the original curves are smooth, the distance between the points on the average curve can be represented by two pixels (the factor 2 results from rounding to integer values). Most of the time the distance can be represented by one pixel or even zero pixels, which occurs when two points are almost on top of each other. (This happens when curves move in opposite directions.) The average curve is represented by a continuous string of pixels that closely approximates the calculated point locations. Redundant pixels are removed by applying Freeman's corner cutter routine (see section 3.1.4).

4.2.3 Results

The average curve's data is stored in the buffer and displayed on the screen. The curves also are assigned a label consisting of a number and the character A (for Average, see Figures 4.15 and 4.16).

Figure 4.8 shows three curves. The curve on the left and right of the figure are the original curves. The one in the center is the average curve that results from unweighted normalized averaging

(see section 4.1.1). Figure 4.9 shows the same two original curves, but the curve in the center is now the curve that results from an unweighted average over the length of the shortest curve. Note the difference between Figures 4.8 and 4.9. Weighted averages of the same original curves are shown in Figures 4.10 and 4.11. The weighting factor for the curve on the left is 1 while the weighting factor for the curve on the right is 1/3, 1, and 3. This results in the average curves that are shown in the middle and that depend on the weighting factor of the curve on the right. The higher the weighting factor for the curve on the right, the more the average curve resembles this curve and the more the average curve's position is shifted towards the curve with the higher weighting factor (see Figure 4.10 and 4.11). The averages shown in Figure 4.10 are obtained from a normalized averaging operation and the averages in Figure 4.11are obtained from averaging each curve over the length of the shortest curve. The routine is very efficient and the results are shown almost instantaneously on the screen (averaging of two frames with ten curves each takes less than one second for a moderately used VAX 11/780 computer). The routine is capable of averaging each curve on a particular frame with the corresponding curves on many other frames. The whole process can be performed in one operation and is shown in Figure 4.12. It enables the user to eliminate disturbances and noise (by averaging) as well as modulation effects that occur over a longer time.

Figures 4.8 through 4.11 show averages of curves taken from actual fluid flows. Figures 4.13 and 4.14 show the top half of a fluid flow image. As shown in Figure 2.1, the bottom half of such an image is usually very similar to the top half. In the actual analysis, both halves have to be included (which is possible with the current version of the program). Finally, Figures 4.15 and 4.16 show the top half of an average frame obtained from averaging the frames shown in Figure 4.13 and 4.14. Figure 4.15 shows the average frame obtained by a normalized length average, while Figure 4.16 shows the average frame that is obtained by a shortest length average.

For ease of viewing, the scaling factor for the curves in Figures 4.8 through 4.11 and 4.13 through 4.16 was set equal to three. This results in displayed curves that contain more pixels than the actual curves which might leave the viewer with the thought that further improvement is possible. However, since all operations use the original data, this is neither possible nor necessary, and the curves are shown with the highest possible resolution.

5. CURVE FILTERING AND RECOGNITION

Filtering or smoothing of curves might be required for a number of reasons. Additional noise or disturbance reduction might be needed or one might want to retrieve the basic or fundamental shape of the curves. For this reason a special smoothing algorithm was developed and this algorithm is presented in the next paragraph.

The second algorithm presented in this chapter calculates a similarity value to express the similarity between a number of curves. This enables the user to calculate how much a curve is changing as a function of time or position. Another application might be to use these similarity values to recognize a frame or curve.

5.1 Curve smoothing

A special smoothing algorithm operating in the spatial domain was developed and is presented in the remainder of this paragraph. The differences between smoothing in the spatial or frequency domain are discussed in the following section.

5.1.1 Smoothing in the spatial versus frequency domain

The discussion in paragraph 2.3 remains valid here. The calculation of Fourier transforms and inverse Fourier transforms is relatively time consuming and costly. Filtering of the curve in the frequency domain is performed by multiplying the curve's transform with the desired filter function. However, the same

operation can be performed in the spatial domain by means of a convolution operation between the curve's function and the inverse transform of the desired filter function (Carlson, 1975). In terms of computation and effort, both operations are comparable (excluding the Fourier transform and inverse Fourier transform operations). So in terms of speed the spatial domain is clearly preferable. The filter operation in the frequency domain will have a dramatic effect on the location of the endpoints of the curves (see Gumas, 1985). The endpoint locations depend on the filtering operation and will be different from the original location. Due to the interchangeability of the convolution operation in the spatial domain and the multiplication operation in the frequency domain, this is also true for the smoothing operation in the spatial domain. However, with a specially developed modification of the convolution operation, it is possible to keep the endpoints fixed. For these reasons, the spatial domain approach is preferred. The developed algorithms are presented in the next section.

5.1.2 Smoothing algorithm

As mentioned in section 3.1.3, the Δx and Δy values for each curve are either -1, 0, or +1. Figure 5.1 shows the Δx and Δy sequences for a particular curve. The curve itself is shown in Figure 5.2. The absolute displacement in the x direction is 4x(-1) +2 x 1 = -2. Since there are eight elements in this curve, the average displacement per element in the x direction is -2/8 = -.25.

Similarly, the absolute and average displacements in the y direction are +4 and .5 respectively. The cumulative averages are also shown in Figure 5.1. The cumulative averages are rounded to the nearest integer. (A value that is exactly between two integer values is rounded to the nearest integer that is smaller than this value.) These rounded cumulative averages are also shown in Figure 5.1. From the latter, the newly smoothed Δx and Δy values are derived (see Figure 5.1). From these values the smoothed curve is derived; it is shown in Figure 5.2. By averaging the Δx and Δy values over the entire length of the curve, all resolution is lost (the average deviation is the same for all elements) and thus no further smoothing is possible. This method is essentially the same as the Freeman (1961) routine that finds the shortest path between the curve's startpoint and endpoint. However, both forms of smoothing produce the same curve for all those original curves that have the same start-point and endpoint (see Figure 5.3) because the cumulative Δx and Δy values are the same for all these curves. This is not always desired. Often one likes to maintain the "fundamental" shape of the curve. Besides, since no convolution operation was involved, it is not really clear what kind of filtering operation was applied. By averaging the Δx and Δy values over the entire curve, each element of the original curve contributed to the value of each element in the smoothed curve. It seems logical that if this range is reduced, the smoothing will also be reduced. This is accomplished by the modified and unmodified convolution operations that are described next.

For a continuous tangent angle function and filter function, the filtering/smoothing operation can be described with a convolution integral (Carlson 1975):

$$S(\ell) = \int_{-A}^{A} \phi(\lambda).f(\ell - \lambda)\, d\lambda \qquad (5.1)$$

where:

$s(\ell)$ = the smoothed function (curve); $s(\ell) = 0$ for $\ell \notin [-(A+W), (A+W)]$

$\phi(\ell)$ = tangent angle function of curve; $\phi(\ell) = 0$ for $\ell \notin [-A,A]$

$f(\ell)$ = filtering/smoothing function; $f(\ell) = 0$ for $\ell \notin [-W,W]$

λ = integration variable

Also $f(\ell)$ is normalized so that:

$$\int_{-W}^{W} f(\ell)\, d\ell = 1 \qquad (5.2)$$

The second equation is needed to obtain unit gain. If the tangent angle function is a constant (straight line) and the filtering function is a constant, then the smoothed value at $\ell = 0$ should have the same tangent angle as the original curve. This is true because equation 5.1 is equivalent to the averaging operation that was described in the beginning of this section. This result is obtained by the following constraint:

$$W \leq A \qquad (5.3)$$

If equations 5.1 through 5.3 are valid, the smoothed curve tangent angle at $\ell = 0$ will be equal to the constant tangent angle of the

original curve. To prove this, consider the following tangent angle function:

$$\phi(\ell) = \begin{cases} \phi \text{ for } \ell \varepsilon [-A,A] \\ o \text{ for } \ell \notin [-A,A] \end{cases} \tag{5.4}$$

Then with equations 5.1 through 5.3, one gets for the smoothed curve:

$$\begin{aligned} S(o) &= \int_{-A}^{A} \phi(\lambda)\ f(o-\lambda)\ d\lambda = \phi \int_{-A}^{A} f(-\lambda)\ d\lambda \\ &= -\phi \int_{A}^{-A} f(\mu)\ d\mu = \phi \int_{-A}^{A} f(\mu)\ d\mu \\ &= \phi \int_{-W}^{W} f(\mu)\ d\mu = \phi .1 = \phi \end{aligned} \tag{5.5}$$

If A = W, the width of the filter and the tangent angle function are the same. This, in fact, is the averaging operation presented at the beginning of this section. For W < A equation 5.5 is still valid, but the width of the filter function is less than the width of the tangent angle function resulting in a local average. So the convolution operation is a local averaging operation.

Figure 5.4 shows the smoothed function when the tangent angle and filter functions are rectangular pulses. Note that both equations 5.2 and 5.3 are valid for these functions. Studying the smoothed function reveals two undesirable effects. First, the "length" of the smoothed curve is longer than that of the original

curve (2A + 2W) and secondly the ends of the tangent angle function of the smoothed curve taper off. The latter results in errors at the end and start of the smoothed curve. This is shown in Figure 5.5 for the functions of Figure 5.4 with $\phi = \pi$ (meaning that the original curve is a horizontal line of length 2A with its startpoint at the right). The shape of the smoothed curve depends on the tangent angle of the original line (compare Figures 5.5 and 5.6).

The reason for these errors is the local averaging operation. The operation averages the original tangent angle function over a fixed region. At the ends of the curve, the tangent angle function is not available anymore over the entire region, while it is still averaged over this fixed region. This causes a gradual drop in the calculated average.

Instead of applying the smoothing operation to the tangent angle function, it is also possible to operate on the x and y coordinate functions of the original curve. This would give:

$$x_s(\ell) = \int_{-A}^{A} x(\lambda)\ f(\ell-\lambda)\ d\lambda \quad \text{and} \quad y_s(\ell) = \int_{-A}^{A} y(\lambda)\ f(\ell-\lambda)\ d\lambda \qquad (5.6)$$

where:

$x_s(\ell)$ = the x coordinate function of the smoothed curve

$y_s(\ell)$ = the y coordinate function of the smoothed curve

$x(\ell)$ = the x coordinate function of the original curve

$y(\ell)$ = the y coordinate function of the original curve

and:

$$x_s(\ell) = 0 \text{ for } \ell \notin [-(A+W),\ (A+W)]$$

$$y_s(\ell) = 0 \text{ for } \ell \notin [-(A+W),\ (A+W)]$$

$$x(\ell) = 0 \text{ for } \ell \notin [-A,A]$$

$$y(\ell) = 0 \text{ for } \ell \notin [-A,A]$$

Equations 5.2 and 5.3 remain valid. So if $x(\ell)$, $y(\ell)$, and $f(\ell)$ are again rectangular pulses, the smoothed functions look similar to $s(\ell)$ in Figure 5.4 and are shown in Figure 5.7. The absolute displacements in the x and y direction (from startpoint to endpoint) for the original curve are equal to:

$$D_{x,old} = \int_{-A}^{A} x(\ell)\ d\ell = 2AX$$

$$D_{y,old} = \int_{-A}^{A} y(\ell)\ d\ell = 2AY \qquad (5.7)$$

where:

$D_{x,old}$ = the absolute displacement of the original curve in the x direction

$D_{y,old}$ = the absolute displacement of the original curve in the y direction

But the absolute displacements for the x and y coordinates of the smoothed curve are (see Figure 5.7):

$$D_{x,new} = \int_{-(A+W)}^{(A+W)} x_s(\ell)\ d\ell = 2AX$$

$$D_{y,new} = \int_{-(A+W)}^{(A+W)} y_s(\ell)\ d\ell = 2AY \qquad (5.8)$$

where:

$D_{x,new}$ = the absolute displacement of the smoothed curve in the x direction

$D_{y,new}$ = the absolute displacement of the smoothed curve in the y direction

So the absolute displacement between the startpoint and endpoint of both the original and smoothed curve is the same. Since both curves have the same startpoint, the endpoints are also the same. This is true in general, as long as equation 5.2 is satisfied.

In the actual implementation, the x and y functions are the discrete Δx and Δy functions that were defined in section 3.1.3. The smoothed x_s and y_s functions will be approximated by the similar discrete functions Δx_s and Δy_s. The equivalent of equation 5.2 becomes:

$$\sum_{k=-W_k}^{W_k} f(k) = 1 \qquad (5.9)$$

The equivalents of equations 5.6 become:

$$x_s^*(k) = \sum_{k=-A_k}^{A_k} \Delta x(n)\ f(k - n) \qquad (5.10)$$

and,

$$y_s^*(k) = \sum_{k=-A_k}^{A_k} \Delta y(n)\ f(k - n) \qquad (5.11)$$

Finally, the equivalent of equation 5.3 becomes:

$$W_k \leq A_k \qquad (5.12)$$

Because $|\Delta x| \leq 1$, $|\Delta y| \leq 1$, and equation (5.9) also $|\Delta x_s^*| \leq 1$ and $|\Delta y_s^*| < 1$. The Δx_s^* and Δy_s^* values represent the displacement in number of pixels and thus have to be integer values. Let Δx_s^* and Δy_s^* be the nearest integer values to x_s^* and y_s^* where the difference between the rounded and original values is carried over to the next sample to avoid cumulative errors. Then Δx_s^* and Δy_s^* are either -1, 0 or +1. So the smoothed curves can be approximated with a continuous string of pixels. An example, with $W_k = 5$, $A_k = 7$, $f(k) = 1/11$ for all k, and $\Delta x(k) = 1$ for $k \varepsilon [-7,7]$ is shown in Figure 5.8. Correcting for cumulative errors causes gaps in the Δx_s^* function in this particular case. This can cause an undesired effect. Consider the data of Figure 5.9. The Δx and Δy are the same as the Δx function in Figure 5.8. So the Δx_s^* and Δy_s^* functions are also the same as the Δx_s^* function of Figure 5.8. Both the original and smoothed curves are plotted in Figure 5.10. The gaps cause no problem here since they occur at the same samples for both the Δx_s^* and Δy_s^* function. If the Δy string is shortened by two samples (one on each side), the gaps in Δy_s^* are shifted by one position (either to the right or to the left depending on which side of the curve they are on. The data of these curves is shown in Figure 5.11. Both curves are shown in Figure 5.12. The gaps do cause a problem this time. The "smoothed" curve is actually less smooth than the original curve. To avoid these problems, the smoothing operation is modified. Consider Figure 5.13.The tapered ends of the $x(\ell)$ and $y(\ell)$ functions are

divided at $\ell = A$ and $\ell = -A$ respectively. The values at the ends of the tapered ends are mirrored with respect to the dividing lines and added to the corresponding original function values. This is shown in Figure 5.13. This process has two advantages. First it guarantees that the Δx_s and Δy_s strings have the same length as the original x and y string. And secondly, the gaps that were present in Figure 5.8 are removed. In fact for the particular Δx of Figure 5.8, Δx and Δx_s are the same. The modified Δx_s and Δy_s have to be rounded to the nearest integer value. Again the difference between the actual and rounded value is carried over to the next sample to avoid cumulative errors. Significant errors can result if this is not done. For example, if the values are continuously rounded to a value that is smaller, the total absolute displacement will not be the same anymore and a different endpoint will result.

Figure 5.14 shows what is established by the modification. In this example $W_k = 2$. The arrows show to which displacement values in the smoothed curve the displacement values of the original curve contribute. The dashed arrows show the folded back contributions. The pixels near the end of the original curve contribute more to the pixels near the end of the smoothed curve. Closer to the end, fewer pixels of the original curve are contributing to the displacement values of the smoothed curve. This, in fact, results in a reduction in smoothing at the ends of the curve. This can be

justified by noting that less information is available for the ends of the original curve. In the middle of the original curve, the shape of the curve is known on both sides of the pixels one is looking at. This is less and less the case for the pixels near the end.

5.1.3 Results

The minimum filter width is one pixel wide ($W_k = 0$). If this filter is used, the original curve is left unchanged. The maximum filter width is constrained by equation 5.12. Maximum smoothing is obtained for $W_k = A_k$. Intermediate values of W_k give different smoothing values. Figures 5.15 through 5.18 show the smoothing of two curves obtained by different values of W_k. The shape of the filtering function was a rectangular pulse, resulting in a sinc (with $\text{sinc } x = \sin(\pi x)/(\pi x)$) filter function in the frequency domain. The routing could be modified for other filters. However, as can be seen from Figures 5.15 through 5.18, excellent smoothing characteristics are obtained, giving the user a lot of flexibility in terms of deciding how much smoothing should be used.

5.2 Similarity calculations

As an aid in the analysis of fluid flows, it would be nice if one could express the similarity between two curves as a numerical value. In the remainder of this paragraph such a technique will be presented.

5.2.1 Similarity calculated over shortest curve versus normalized length

Basically the discussion in paragraph 4.1.1 also applies to this case. So again the user is given the option to calculate the similarity value over either the full lengths of the curves (which then have to be normalized to obtain an equal amount of measurement points per curve) or over the length of the shortest curve. This choice is decoupled from the choice of either method in the averaging algorithm for two reasons. First, one might want to measure the similarity value between two of the original curves. Second, if an average curve was obtained from a set of original curves by using the shortest length method, one might still want to compare one of the complete original curves with the average curve.

5.2.2 Cross-correlation measurements

At first glance, a cross-correlation measurement between two curves will give the desired similarity value. Suppose we have two curves with the tangent angle functions $\phi_1(\ell)$ and $\phi_2(\ell)$. The cross-correlation value (Carlson 1975) between these two curves is equal to:

$$R_{\phi_1\phi_2}(\tau) = \int_{-A}^{A} \phi_1(\ell)\ \phi_2^*(\ell-\tau)\ d\ell = \int_{-A}^{A} \phi_1(\ell)\ \phi_2(\ell-\tau)\ d\ell \qquad (5.13)$$

where

$\phi_2^*(\ell)$ = the complex conjugate of $\phi_2(\ell)$ which is equal to $\phi_2(\ell)$ because $\phi_2(\ell)$ (and $\phi_1(\ell)$) is a real function

$\phi_1(\ell) = 0$ for $\ell \notin [-A,A]$

$\phi_2(\ell) = 0$ for $\ell \notin [-A,A]$

τ = a shift operator

Now suppose $\phi_1(\ell) = -\frac{\pi}{2}$ and $\phi_2(\ell) = -\frac{\pi}{4}$ for $\ell \varepsilon [-A,A]$. Their $R_{\phi_1\phi_2}(0) = A\pi^2/4$. If $\phi_3(\ell) = -\frac{3\pi}{4}$ for $\ell \varepsilon [-A,A]$, then $R_{\phi_1\phi_3}(0) = 3A\pi^2/4$. So $R_{\phi_1\phi_2}(0) = 3R_{\phi_1\phi_3}(0)$. The three curves are shown in Figure 5.19. This result is undesirable. $R_{\phi_1\phi_2}(0)$ should equal $R_{\phi_1\phi_3}(0)$ because in fluid flows the difference between ϕ_2 and ϕ_1 is equal to the difference between ϕ_3 and ϕ_1. Also if the camera in the set-up of Figure 2.1 is rotated over 180^o, the timelines propagate from right to left and the fluid flow is a mirrored image of the one shown in Figure 2.1. This is shown for curves ϕ_1 and ϕ_2 in Figure 5.20. The mirrored image of curve $\phi_1(\phi_1')$ is equal to its original while the mirrored image of curve $\phi_2(\phi_2')$ is equal to curve ϕ_3 in Figure 5.19. So a simple rotation of the camera would give different similarity value if this method is used, which is undesirable.

Instead of using the horizontal as the line of reference for measuring the curve angles, one can use a vertical line. This is shown in Figure 5.21. Now $\phi_1(\ell) = 0$, $\phi_2(\ell) = \frac{\pi}{4}$ and $\phi_3(\ell) = -\frac{\pi}{4}$ for $\ell \varepsilon [-A,A]$. So both $R_{\phi 1\phi 2}(0)$ and $R_{\phi 1\phi 3}(0)$ are equal to zero. Both result in the same value, but so would any other correlation involving curve ϕ_1. So no matter what the shape of the other curve is, if it is correlated with ϕ_1, the result will be zero.

This is clearly undesirable. The solution seems to lie in the normalization of equation 5.13. To do so, equation 5.13 is divided by the norm of both ϕ_1 and ϕ_2. Thus

$$R^*_{\phi_1\phi_2}(\tau) = \frac{\int_{-A}^{A} \phi_1(\ell)\ \phi_2(\ell-\tau)\ d\ell}{||\phi_1||.||\phi_2||}$$

$$= \frac{\int_{-A}^{A} \phi_1(\ell)\ \phi_2(\ell-\tau)\ d\ell}{\sqrt{\int_{-A}^{A} |\phi_1(\ell)|^2 d\ell \cdot \int_{-A}^{A} |\phi_2(\ell)|^2 d\ell}}$$

$$= \frac{\int_{-A}^{A} \phi_1(\ell)\ \phi_2(\ell-\tau)\ d\ell}{\sqrt{\int_{-A}^{A} \phi_1^2(\ell)\ d\ell \cdot \int_{-A}^{A} \phi_2^2(\ell)\ d\ell}} \qquad (5.14)$$

and with Schwarz's inequality:

$$|R^*_{\phi_1\phi_2}(\tau)| \leq 1 \qquad (5.15)$$

So with the same ϕ_1, ϕ_2, and ϕ_3 as before, one now obtains $||\phi_1|| = \sqrt{2A} \cdot \pi/2$, $||\phi_2|| = \sqrt{2A} \cdot \pi/4$, and $||\phi_3|| = \sqrt{2A} \cdot 3\pi/4$. And thus $R^*_{\phi_1\phi_2}(0) = 1$ and $R^*_{\phi_1\phi_3}(0) = 1$. So both values are equal as desired. However, any two angles that are not equal to zero would have given the same result. So for the curves in Figure 5.22, both $R^*_{\phi_1\phi_2}(0)$ and $R^*_{\phi_1\phi_3}(0)$ are equal to one. Different values for R^* are only obtained if the general shape of both curves

is different. Since both are straight lines of the same length, the value for R^* is always one. Again, this is not a desired characteristic. In terms of fluid flows the similarity between ϕ_1 and ϕ_2 of Figure 5.22 should be different from the similarity between ϕ_1 and ϕ_3. So a different similarity function had to be developed. This function is presented in the next section.

5.2.3 Similarity function

Similar to the technique described in section 4.2.1, each curve is chopped up in an equal number of elements all of the same length. Again, either the full length of the curves or only a part of the curve with a length equal to the shortest curve is used. It is now possible to compare the direction of corresponding elements on each curve. The difference in angle between one element of each curve is expressed as a numerical value. In analogy with equation 5.15 this value is normalized to a value between -1 and +1. Figure 5.23 shows two curves consisting of seven elements each. The element similarity function is defined as:

$$A^*(k) = 1 - 2.\frac{|\phi_1(k) - \phi_2(k)|}{180^o} \qquad (5.16)$$

where

$A^*(k)$ = similarity between elements k of curve 1 and 2. This function is plotted in Figure 5.24. If the element of curve 1 ($\phi_1(\ell)$) is used as reference (pointing straight up in Figure 5.24), a point of the element of curve 2 ($\phi_2(\ell)$) lies anywhere on the circle.

The element similarity values are linearly distributed over this circle and a few of them are shown. In particular the value is equal to 1 if the elements have the same direction, -1 if they are pointing in opposite directions (note each element has a startpoint and endpoint that depends on the direction of the curve), and 0 if they are perpendicular. By summing all these values for all elements, one ends up with a value between -N and +N where N is the number of elements. The normalized similarity value is now defined as:

$$S^*_{\phi_1\phi_2} = \frac{\sum_{k=1}^{N} A^*(k)}{N} \qquad (5.17)$$

where:

$S^*_{\phi_1\phi_2}$ = the normalized similarity value

and $S^*_{\phi 1\phi 2}$ has a value between -1 and +1. Consider again the three curves in Figure 5.19. Then $S^*_{\phi_1\phi_2} = S^*_{\phi_1\phi_3} = 0.5$ and for the curves in Figure 5.22 one obtains $S^*_{\phi_1\phi_2} = 0.75$ and $S^*_{\phi_1\phi_3} = 0.5$. So, as desired, this function expresses the difference between curves ϕ_2 and ϕ_3 in Figure 5.22, while it also expresses the similarity between curves ϕ_2 and ϕ_3 in Figure 5.19. Now consider the two jagged curves of Figure 5.25. For these curves (both with ten elements) $S^*_{\phi_1\phi_2} = 0$, while the curves are in fact very similar. The zero value results from the fact that each element in curve 1 is perpendicular to the corresponding elements in curve 2. If one would shift either of the curves one element up or down, a value of $S^*_{\phi_1\phi_2} = .9$ would result. The .9 results from the fact that now one element of each curve is not contributing to

the final result any longer while the other elements are all aligned. This result is also shown in Figure 5.25. If one would define the number of elements equal to 9 for this case, $S^{*}_{\phi_1\phi_2}$ would equal 1. However, this is not done to express the slight difference between this case and the one where the elements of the two jagged curves would have lined up. To enable this shift a modified similarity function is defined as:

$$S'_{\phi_1\phi_2}(\tau) \triangleq \frac{\sum_{k=1}^{N} A(k,\tau)}{N} \tag{5.18}$$

where:

$$A(k,\tau) \triangleq \begin{cases} 1 - 2.\dfrac{|\phi_1(k) - \phi_2(k-\tau)|}{180^{o}} & \text{for } (k-\tau)\varepsilon[1,N] \\ 0 & \text{for } (k-\tau)\varepsilon[1,N] \end{cases} \tag{5.19}$$

τ = the shift operator (number of elements shifted)

Finally, consider Figure 5.26. Shown are three curves with different lengths of which the first two have the same general shape. After normalization the similarity between curves 1 and 3 or between curves 2 and 3 would result in the same value. Also $S'_{\phi_1\phi_2}(0) = 1$. It would be desirable if the difference in length could also be expressed in the similarity value since in terms of fluid flows, curve 1 and 2 of Figure 5.26 are not the same. To do so, $S'_{\phi_1\phi_2}(t)$ is multiplied with the length of the shortest curve

(L_1 here) divided by the geometric mean of the length of the two curves (L, L_2 here). So the new similarity function is defined as:

$$S_{\phi_1\phi_2}(\tau) \triangleq S'_{\phi_1\phi_2}(\tau)\frac{L_1}{\sqrt{L_1 L_2}} = \frac{\sum_{k=1}^{N} A(k,\tau)}{N} \cdot \sqrt{\frac{L_1}{L_2}} \qquad (5.20)$$

where

L_1 = the length of the shortest curve

L_2 = the length of the longest curve

Note that still

$$|S_{\phi_1\phi_2}(\tau)| \leq 1 \qquad (5.21)$$

The equal sign is only valid if two curves are exactly the same (both in shape and length).

The justification for this particular modification is given in terms of an example. Figure 5.27 shows two semicircles, one having twice the radius of the other. The similarity value is now calculated over the length of the shortest curve. Since the length of the larger semicircle is twice the length of the smaller one, only half of this semicircle contributes to the similarity value. Suppose the first part of each semicircle with length L is approximated with one element (see Figure 5:28). Use of equation 5.20 and 5.19 with $L_1 = L_2 = L$ gives a similarity value of:

$$^1S_{\phi_1\phi_2}(0) = \frac{1 - 2.\frac{45^o - 0^0}{180^o}}{1} = 0.5$$

Approximating the same parts of the semicircles with two elements (see Figure 5.29) gives a value of:

$$^{2}S_{\phi_1\phi_2}(0) = \frac{(1 - 2.\frac{67.5 - 45}{180}) + (1 - 2.\frac{67.5}{180})}{2} = \frac{0.75 + 0.25}{2} = .5$$

which is the same value as before. If the curves are approximated with infinitesimal small elements, the following result is obtained (see Figure 5.30). The angle ϕ_1 at a particular point on the larger semicircle is equal to (see Figure 5.30):

$$\phi_1(\ell) = 90^{o} - \alpha(\ell)$$

where

$$\alpha(\ell) = \frac{\ell}{L}.90^{o}$$

and ℓ = the distance measured along the curve from the point one is looking at to the startpoint of the curve; $\ell\varepsilon[o,L]$

Similarly,

$$\phi_2(\ell) = 90^{o} - \beta(\ell)$$

where

$$\beta(\ell) = \frac{\ell}{L}.180^{o}$$

and thus

$$|\phi_1(\ell) - \phi_2(\ell)| = \frac{\ell}{L}.90^{o} \qquad (5.22)$$

If the summation in equation 5.20 is replaced with an integral, N is replaced by $L/d\ell$, $L_1 = L_2 = L$ and equation 5.19 is replaced with its continuous equivalent, then one gets with equation 5.22:

$$^{3}S_{\phi_1\phi_2}(0) = \int_0^L \frac{1 - 2.\frac{\frac{\ell}{L}.90^{o}}{180^{o}}}{L \,/\, d\ell} \sqrt{\frac{L}{L}} = \frac{1}{L}\int_0^L (1 - \frac{\ell}{L})\, d\ell$$

$$= \frac{1}{L}\left(\ell - \frac{\ell^2}{2L}\right)\Bigg|_0^L = \frac{1}{L}\left(L - \frac{L^2}{2L}\right) = .5 \qquad (5.23)$$

Note that the square root term did not affect the final value. So again the same result is obtained. Now consider Figure 5.31. The similarity value (using equation 5.20) for curves 1 and 2 is 0.5 (see equation 5.23). If one compares curve 1 with itself, the similarity value is 1. Finally, the similarity value for curves 1 and 3 (using equation 5.20 and normalizations) gives the value $\sqrt{\frac{1}{2}} = \frac{1}{2}\sqrt{2} \simeq 0.71$. Physically, curve 3 is less similar to curve 1 than curve 1 itself is (because their lengths differ), but is definitely more similar to curve 1 than curve 2 is. The obtained values express this difference (values of 0.71, 1, and 0.5 were obtained respectively) and the additional term in equation 5.20 is thereby justified. Of course, the term only influences the measurements on normalized curves.

The original flow (see Figure 2.1) shows significant symmetry between the bottom and top half of the flow. However, equation 5.20 cannot directly be used to measure the symmetry. The general direction of the curves is opposite to each other, but the individual elements are not opposite to each other. This is shown in Figure 5.32. If one of the curves is mirrored with respect to a vertical

line, perfect symmetry with respect to a horizontal line would yield that the individual elements of curve 1 and the corresponding elements of the mirrored image of the other curve 1' would always point in opposite directions (see Figure 5.32). This in turn results in a similarity value of -1. Less than perfect symmetry would result in a value larger than -1 (but less than +1). This option is provided to the user, so he can measure the symmetry of the fluid flow. The user also can specify a shift range over which $S_{\phi_1\phi_2}(\tau)$ is calculated. The algorithm will return the maximum magnitude value that was found in this range. So finally the similarity between two curves is defined as:

$$\text{Similarity between 2 curves} = \text{sign}\{S_{\phi_1\phi_2}(\tau)\} \cdot \max_{\tau} \left| S_{\phi_1\phi_2}(\tau) \right| \quad (5.24)$$

where:

$\tau \epsilon [-R, R]$

R = shift parameter (integer value equal to the number of pixels one wants to shift curves over)

$S_{\phi_1\phi_2}(\tau)$ = defined in equation (5.20)

If more than two curves have to be averaged, the algorithm will return all the individual values (obtained from the comparisons of two curves), the mean of these values, and the standard deviation of the set of values.

5.2.4 Results

The similarity measurements shown in Figures 5.33 through

5.36 and Figure 5.38 are obtained by using the normalized length option while the measurement in Figure 5.37 is obtained by using the shortest curve length option.

The similarity value resulting from a comparison between a curve with itself is shown in Figure 5.33. Figure 5.34 shows a similarity measurement between two different curves while Figure 5.35 shows a measurement involving three curves. Shown are the mean value of the two measurements involved and the standard deviation. The number of curves could be increased to the maximum number that can be displayed on the screen which is constricted by the size of the buffer (see paragraph 3.3). The user can pick any curve that is displayed on the screen (original curve, smoothed curve, and averaged curve) with the special cursor routine (see paragraph 3.4) and the similarity routine will retrieve all the required curve data from the stack.

Figure 5.36 shows a similarity measurement between the same two curves of Figure 5.34, but this time a shift range equal to 20% of the length of the curve was specified. The routine returns the maximum similarity value it finds in this range. Figure 5.37, again shows the similarity between the curves of Figure 5.34 (with zero shift), but this time the shortest curve length option was used in the calculation. Finally, Figure 5.38 shows the similarity between a curve and the smoothed curve that was obtained by a maximum smoothing operation on the same original curve. A similarity

value of 1 should result if the curves are the same (Figure 5.33), a value of -1 results if the curves have the same shape and length but their directions are shifted over 180^{o}, and a value of zero should result if the curves are totally dissimilar (averaging out of the individual element values will occur in this case). This is established by the algorithm presented in this chapter. It is capable of measuring the similarity between arbitrary curves and frames.

6. CONCLUSIONS

The feasibility of image processing and pattern recognition of flow patterns has been shown in this investigation. With the aid of the developed algorithms and routines, the user can perform averaging, smoothing and similarity operations on arbitrary curves and frames. With a specially developed algorithm, it is possible to express the similarity between two curves in a numerical value.

The developed code provides a highly interactive program. The program is very flexible and provides the user with many options. He can process complete frames and curves at once to speed up the process, or he can perform operations on just a single curve which gives him maximum flexibility. The program is user friendly, practically menu driven, and almost "idiot proof" in terms of that it rejects inconsistent data inputs.

The algorithm and routines are very efficient in the sense that maximum attention was given to computational efficiency and use of the most effective theories. Averaging and smoothing of curves was established without the use of discrete Fourier transforms. All of this also results in relatively fast routines where the results (even if a large number of frames or curves is involved) are available almost instantaneously. This in turn increases the interactive use of the program.

Efficient data storage and manipulation is achieved by means of specilly developed buffer and cursor routines. These routines form an integral part of the whole program.

Future work should include the integration of the program with the data acquisition and preprocessing equipment and routines. As a special feature, the user might be provided with an option in which a set of previously defined manipulations is operating on the frames automatically. The capabilities of the program (in terms of the maximum number of frames, the maximum number of curves per frame, and the maximum number of points per curve) can be suited to the user's need by changing the appropriate dimension statements.

Of course, the application of the program in fluid flow analysis is one of the immediate future goals.

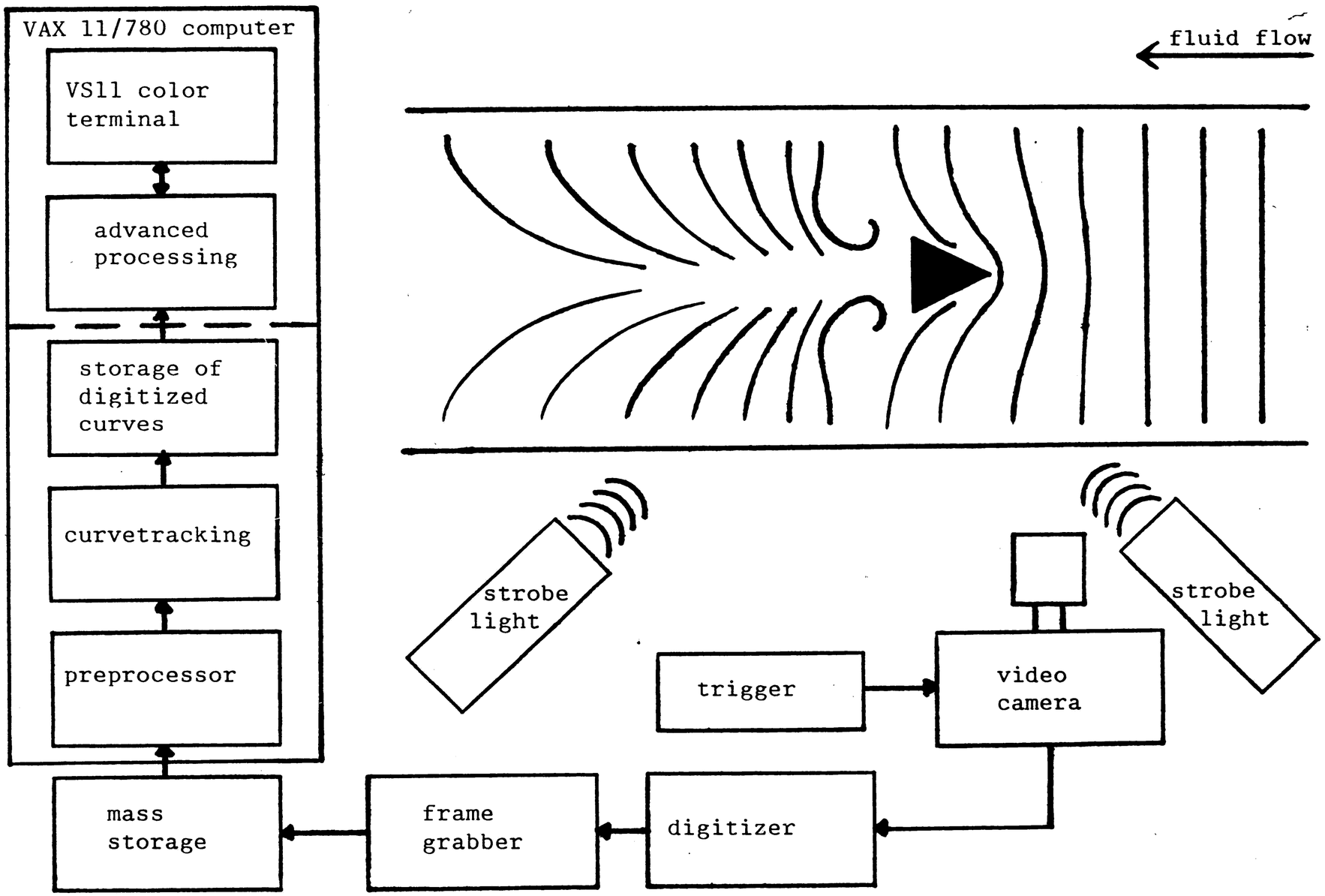


Figure 2.1 Fluid flow analysis setup.

RETAKE

The Operator has Determined that the Previous Frame is Unacceptable and Has Refilmed the Page in the Next Frame.

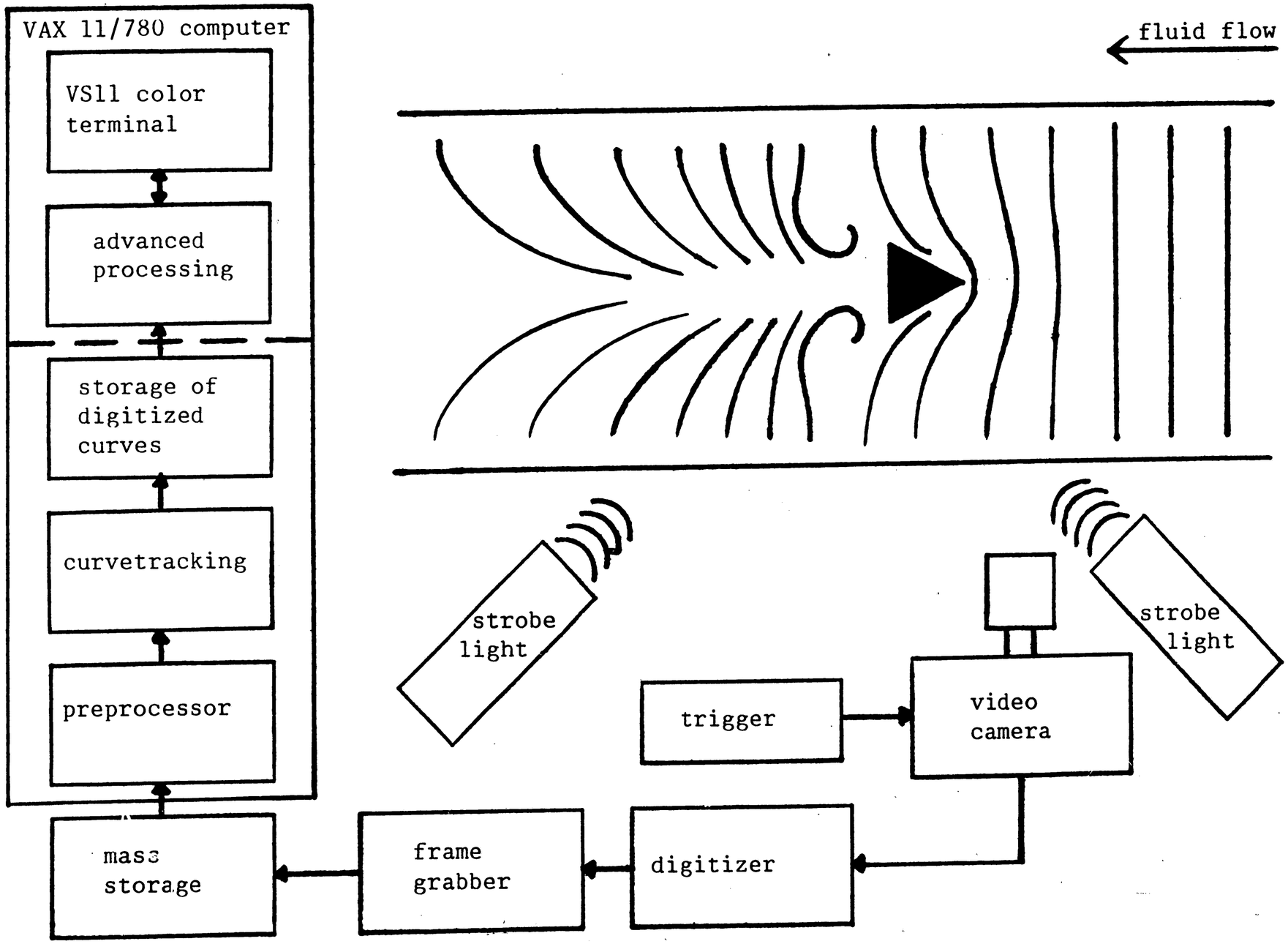


Figure 2.1 Fluid flow analysis setup.

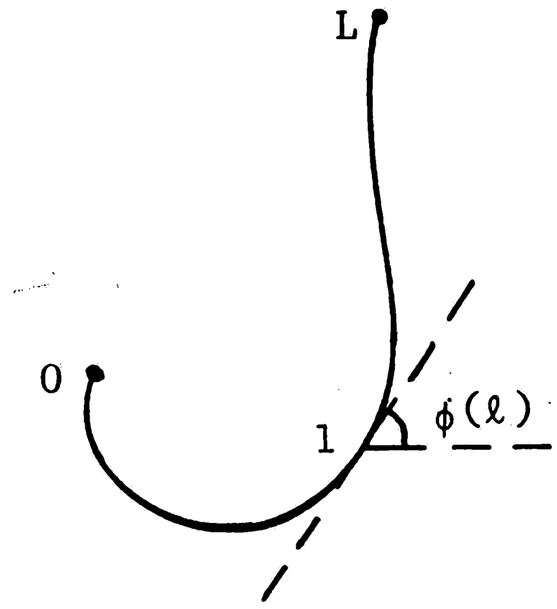


Figure 2.2 Uniform continuous curve $\phi(\ell)$.

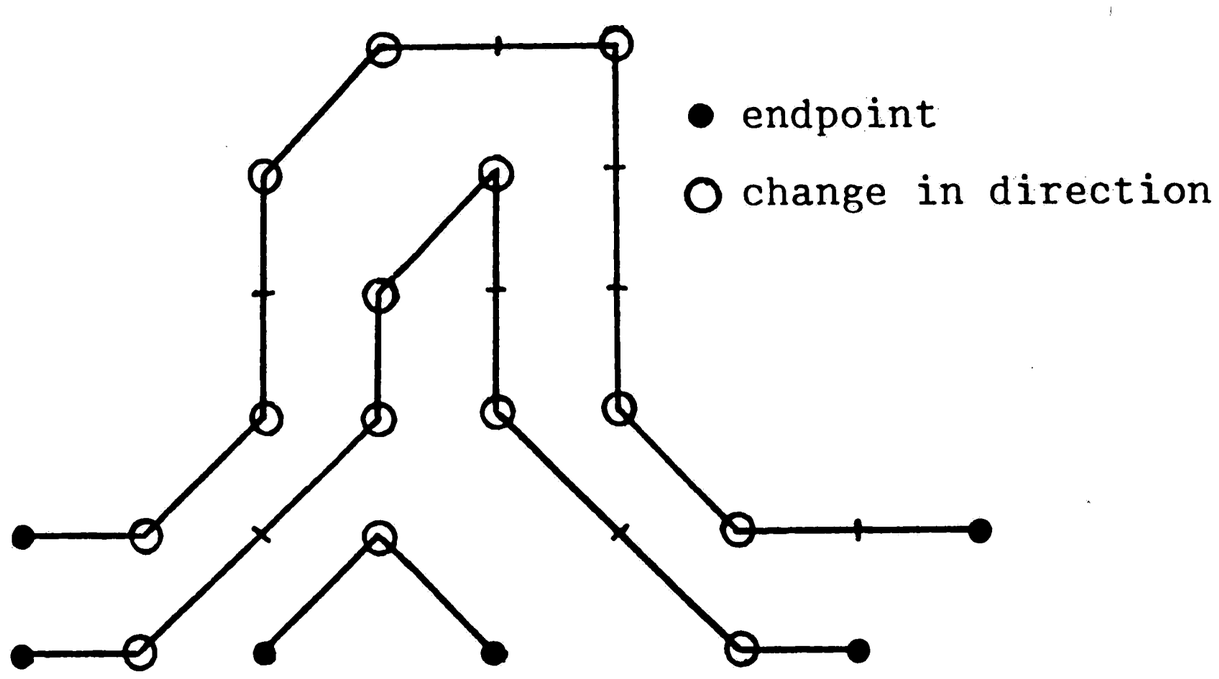


Figure 2.3 Feature extraction in fingerprints.

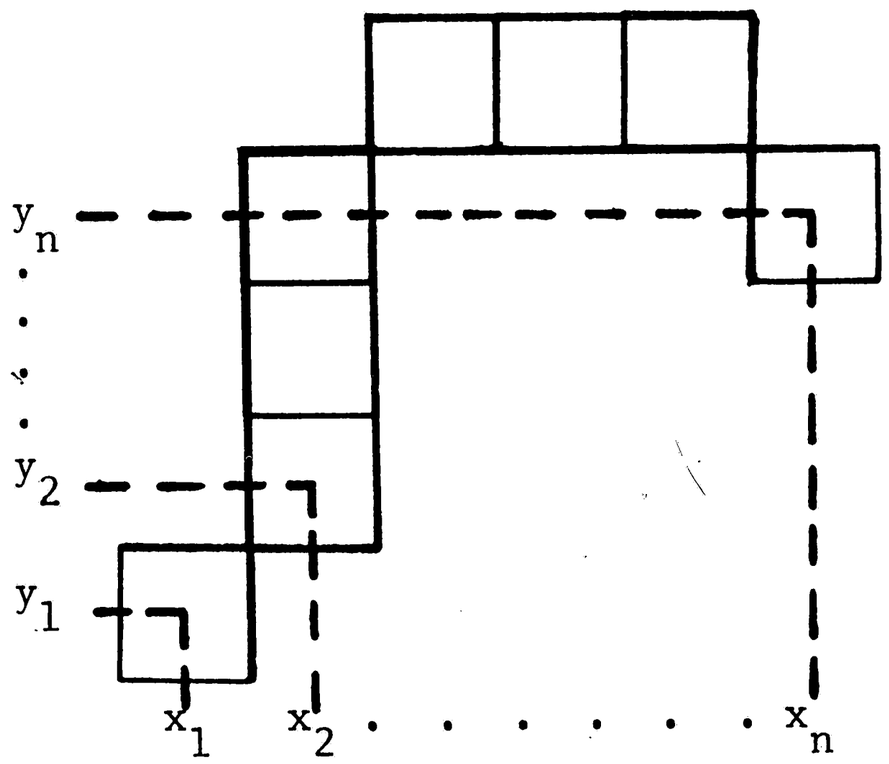


Figure 3.1 Recording x and y coordinate of each pixel.

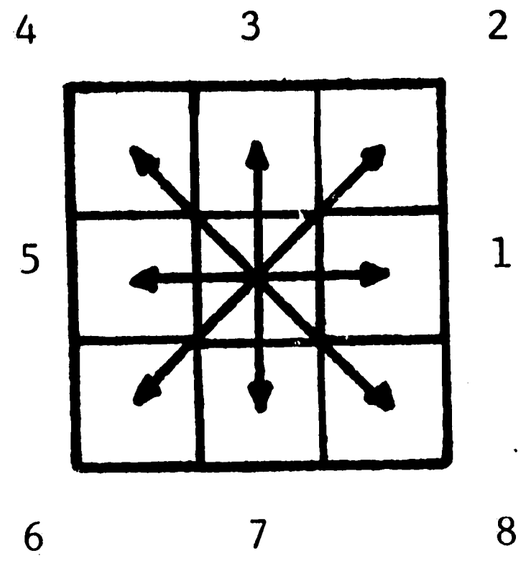
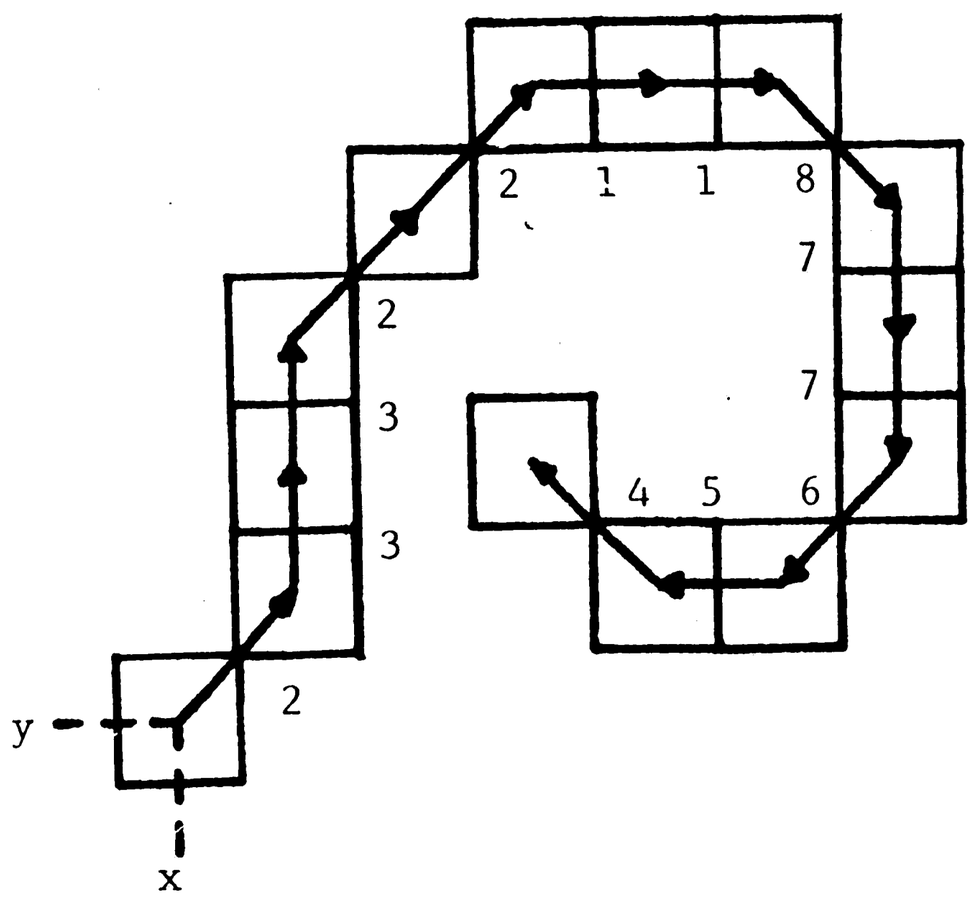


Figure 3.2 Chain code representation.



start: (x,y)
chain codes: 2
3
3
2
2
1
1
8
7
7
6
5
4

Figure 3.3 Curve representation with chain codes.

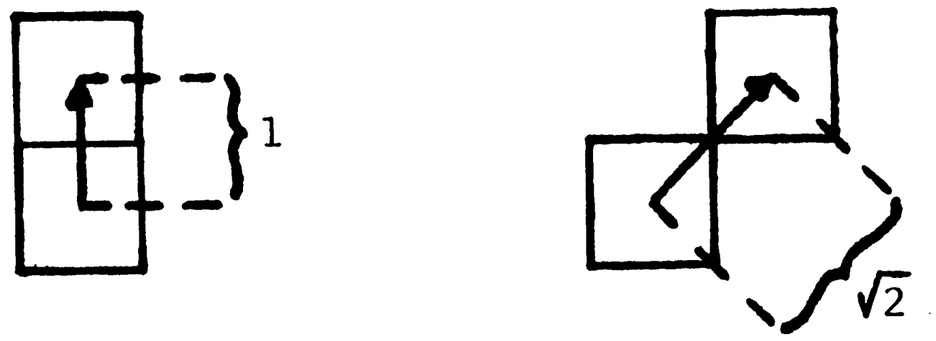
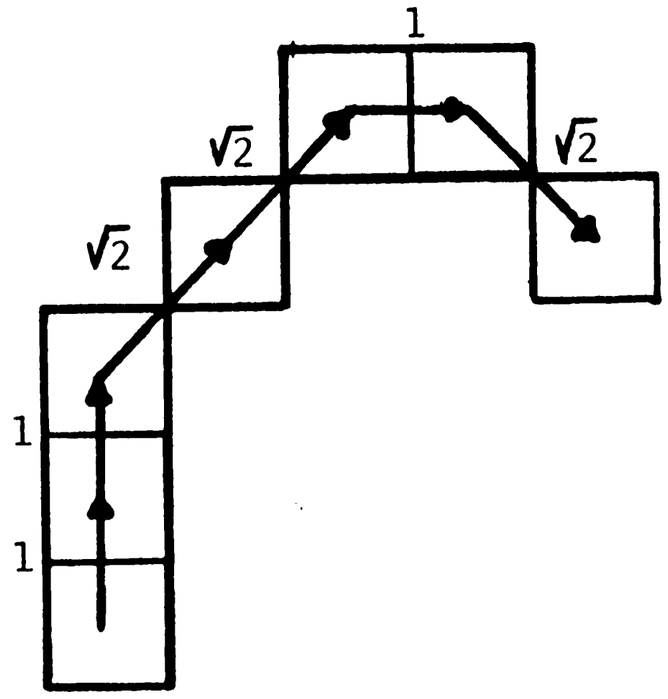
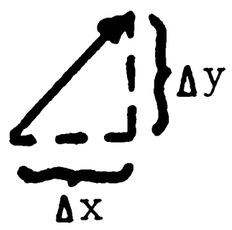


Figure 3.4 Element length.



$L(1) = 1$
$L(2) = 2$
$L(3) = 2 + \sqrt{2}$
$L(4) = 2 + 2\sqrt{2}$
$L(5) = 3 + 2\sqrt{2}$
$L(6) = 3 + 3\sqrt{2}$
$L(7) = 0$

Figure 3.5 Length array data.



chain code value	Δx	Δy
1	1	0
2	1	1
3	0	1
4	-1	1
5	-1	0
6	-1	-1
7	0	-1
8	1	-1
9	0	0
0	end of curve	

Figure 3.6 Chain code to Δx/Δy conversion.

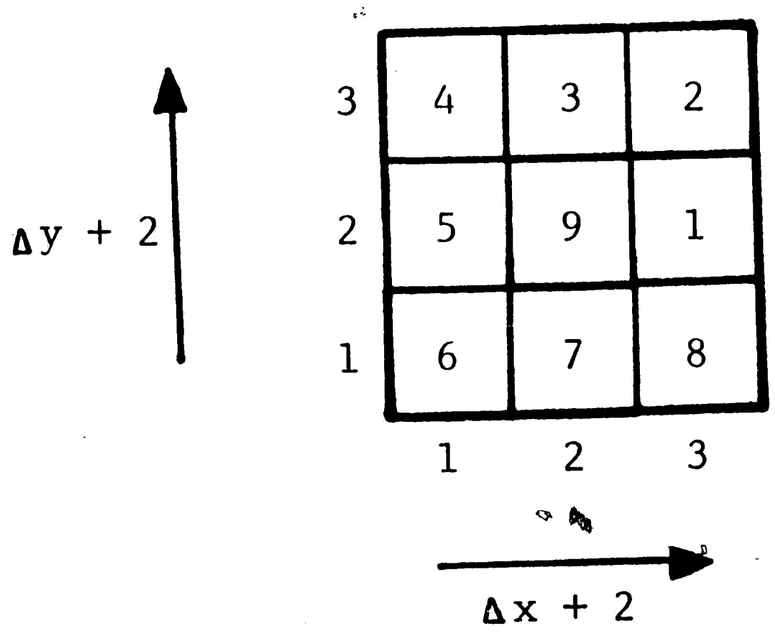


Figure 3.7 Δx/Δy to chain code conversion.

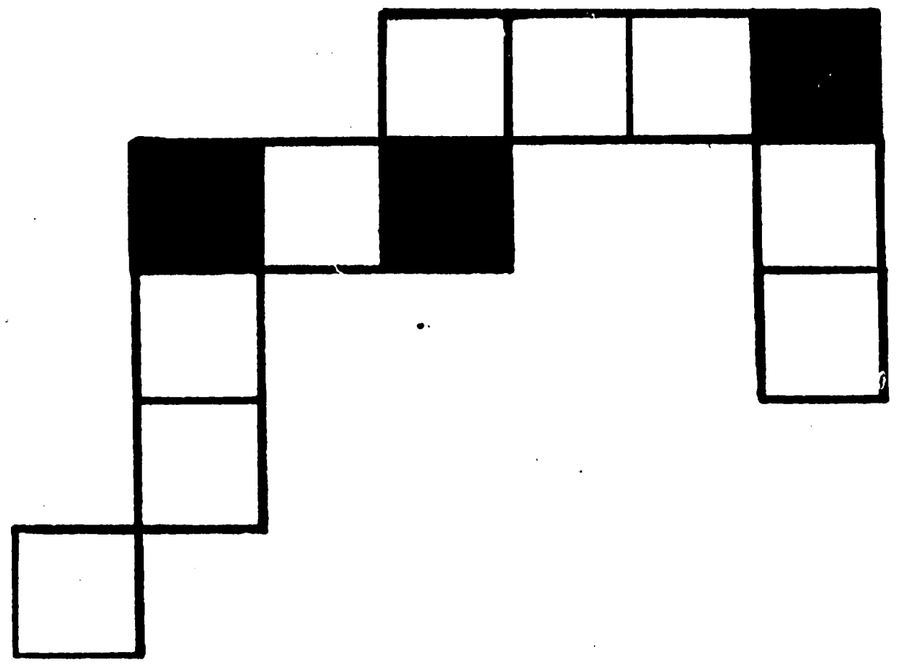


Figure 3.8 Removing redundant pixels.

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

Figure 3.9 Freeman's corner cutter matrix.

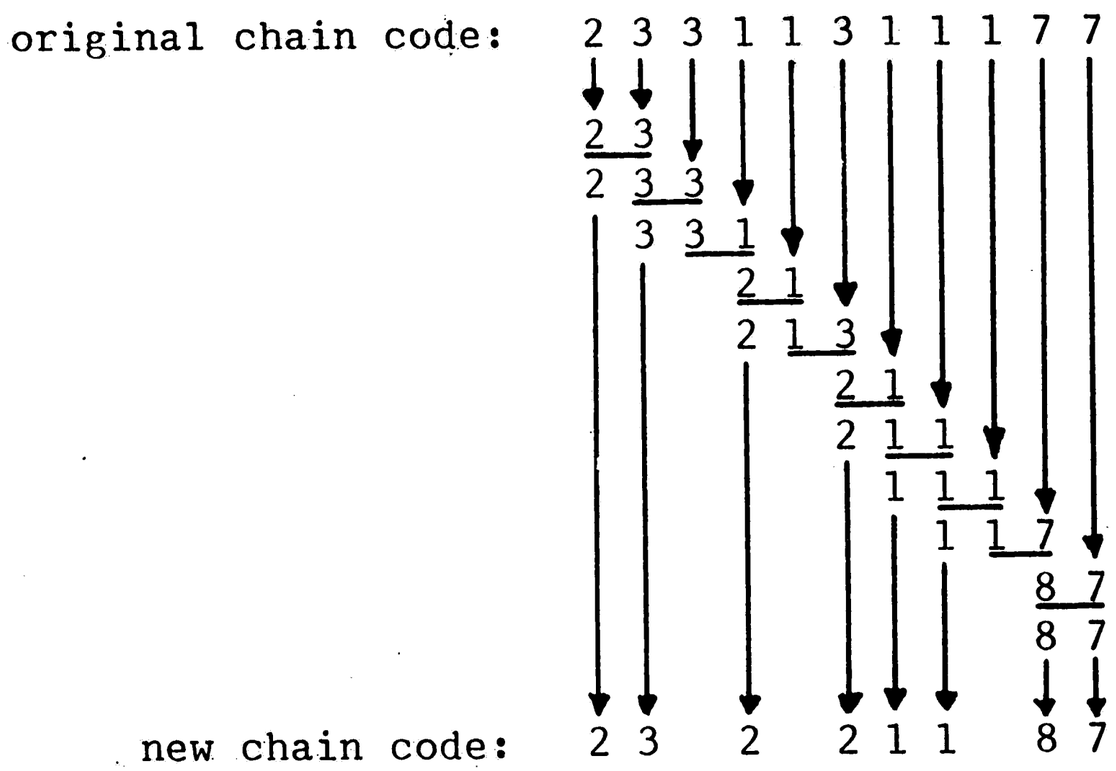


Figure 3.10 Freeman's corner cutter applied to the curve of Figure 3.8.

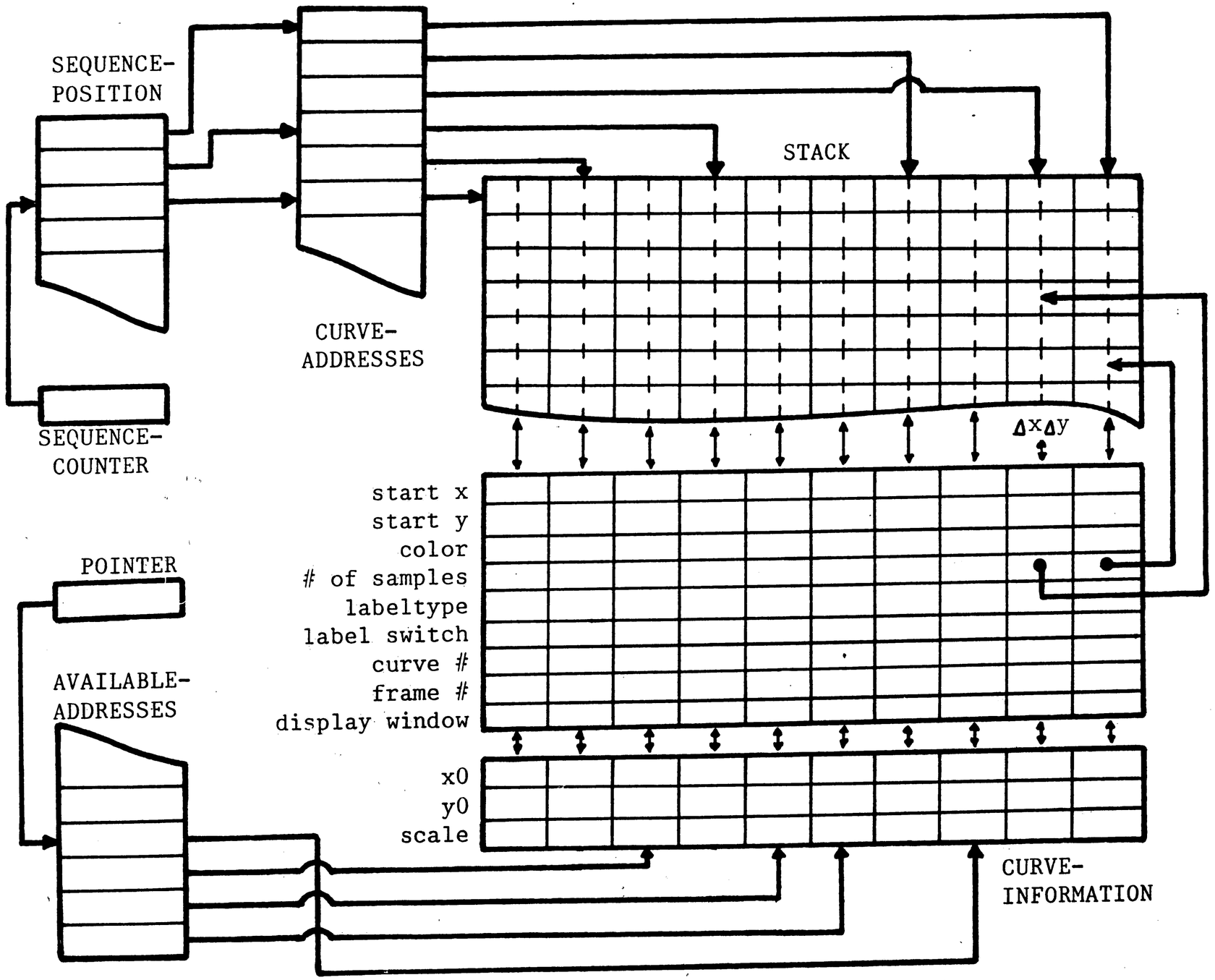


Figure 3.11 Buffer setup.

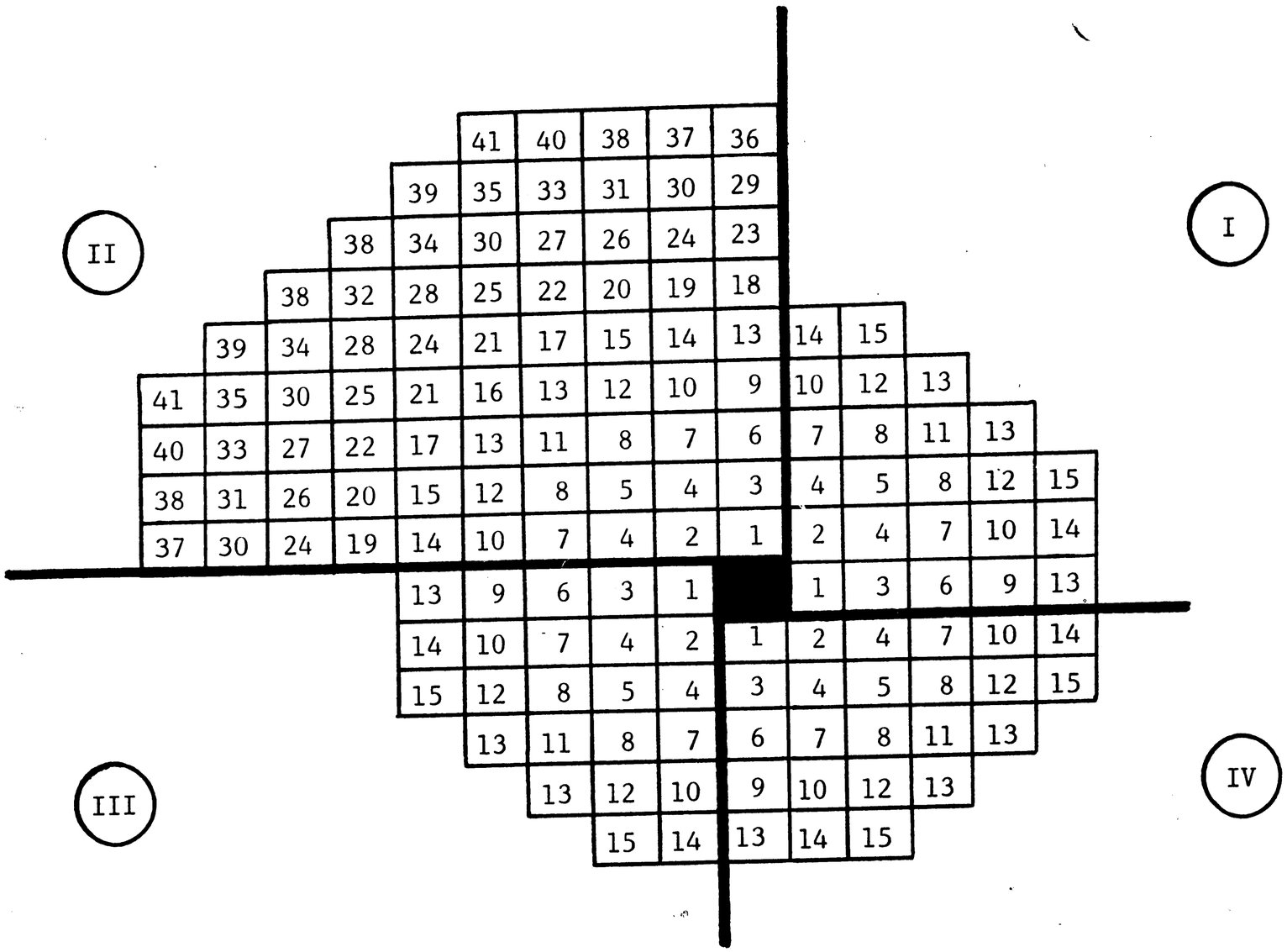


Figure 3.12 Cursor search routine.

Pixel #	Quadrant			
	II	III	IV	I
1	0,1	-1,0	0,-1	1,0
2	-1,1	-1,-1	1,-1	1,1
3	0,2	-2,0	0,-2	2,0
4 (a)	-1,2	-2,-1	1,-2	2,1
4 (b)	-2,1	-1,-2	2,-1	1,2

Figure 3.13 Search (look-up) table.

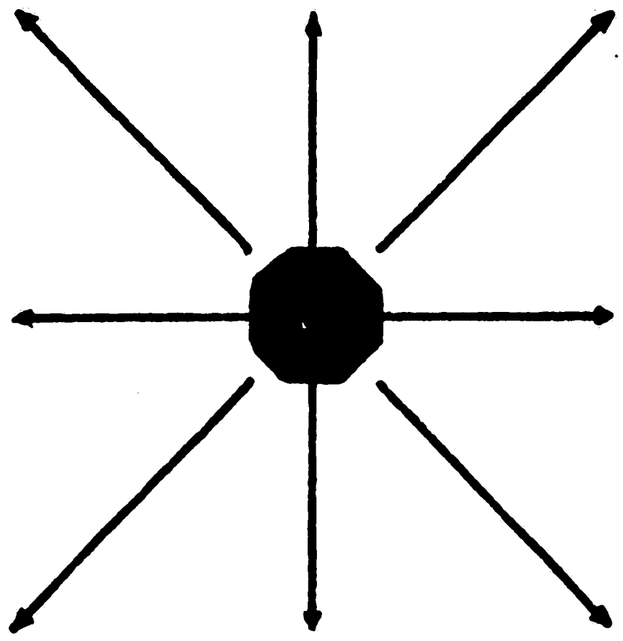


Figure 3.14 Expanded search routine.

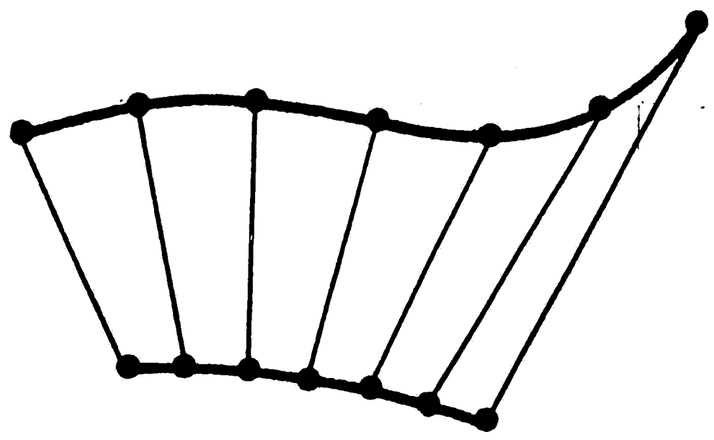


Figure 4.1 Normalized length.

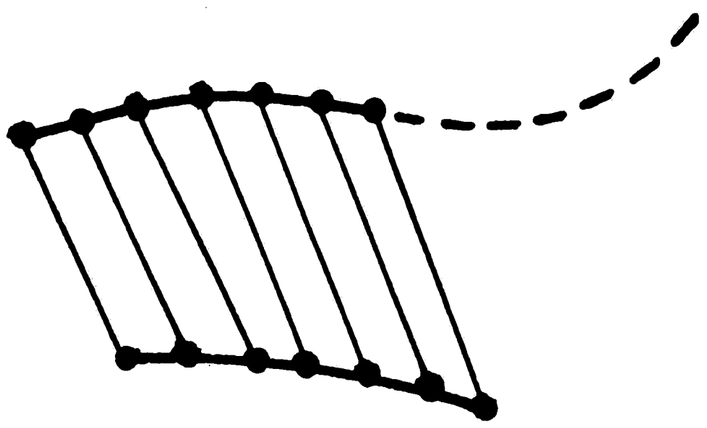


Figure 4.2 Shortest curve length.

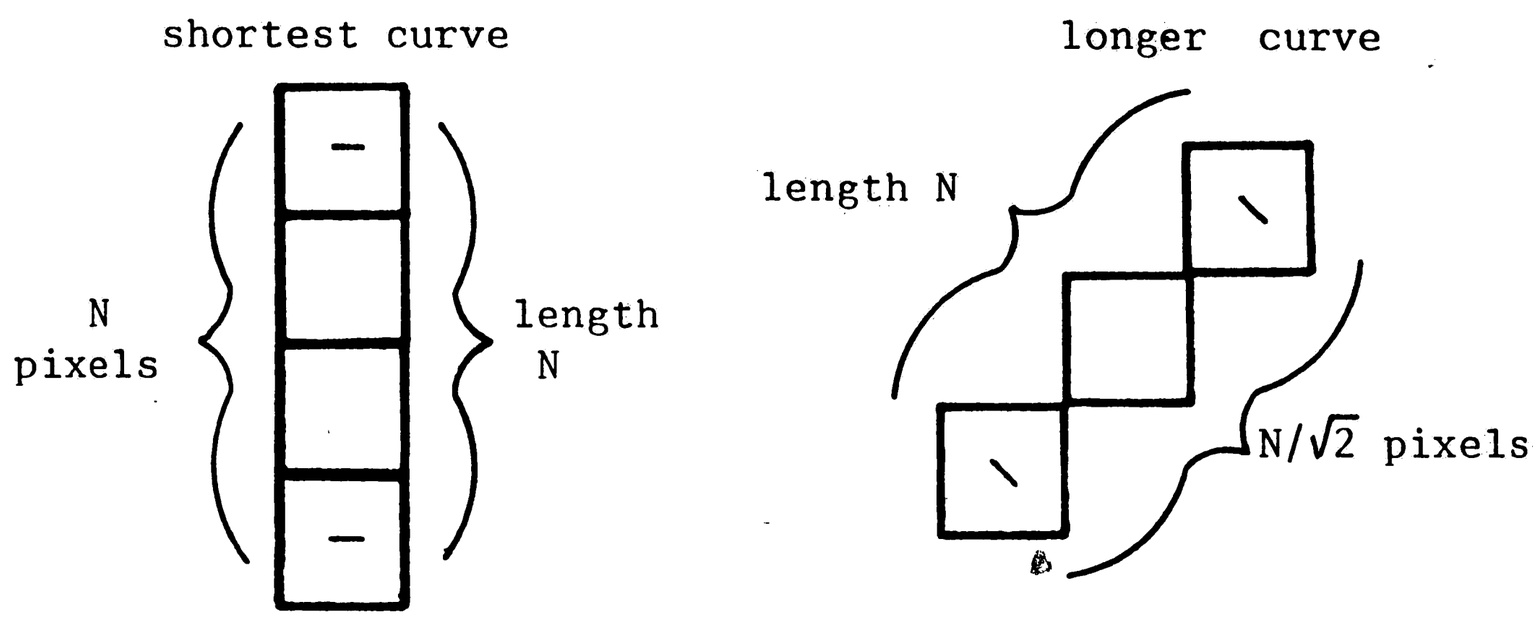


Figure 4.3 Minimum number of pixels in second curve.

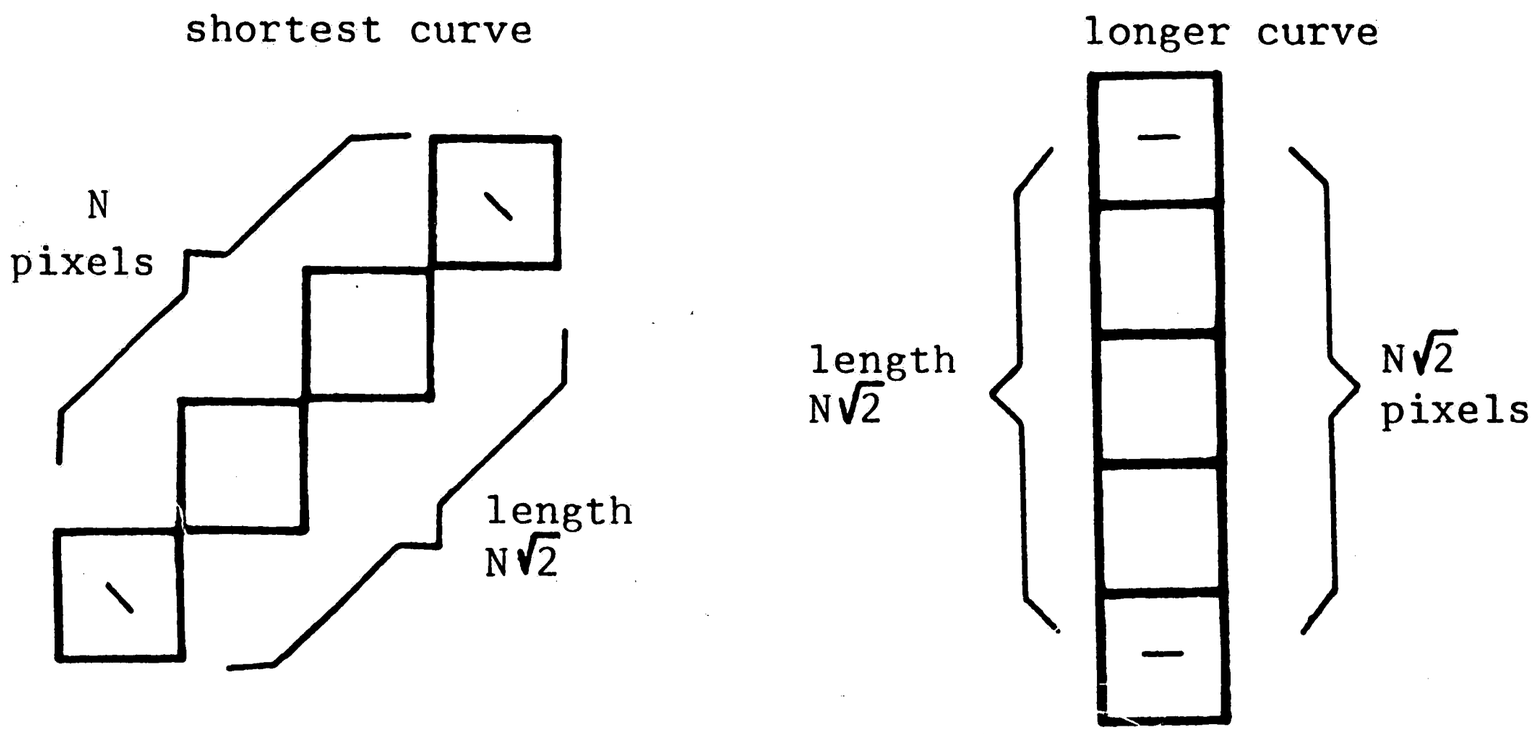


Figure 4.4 Maximum number of pixels in second curve.

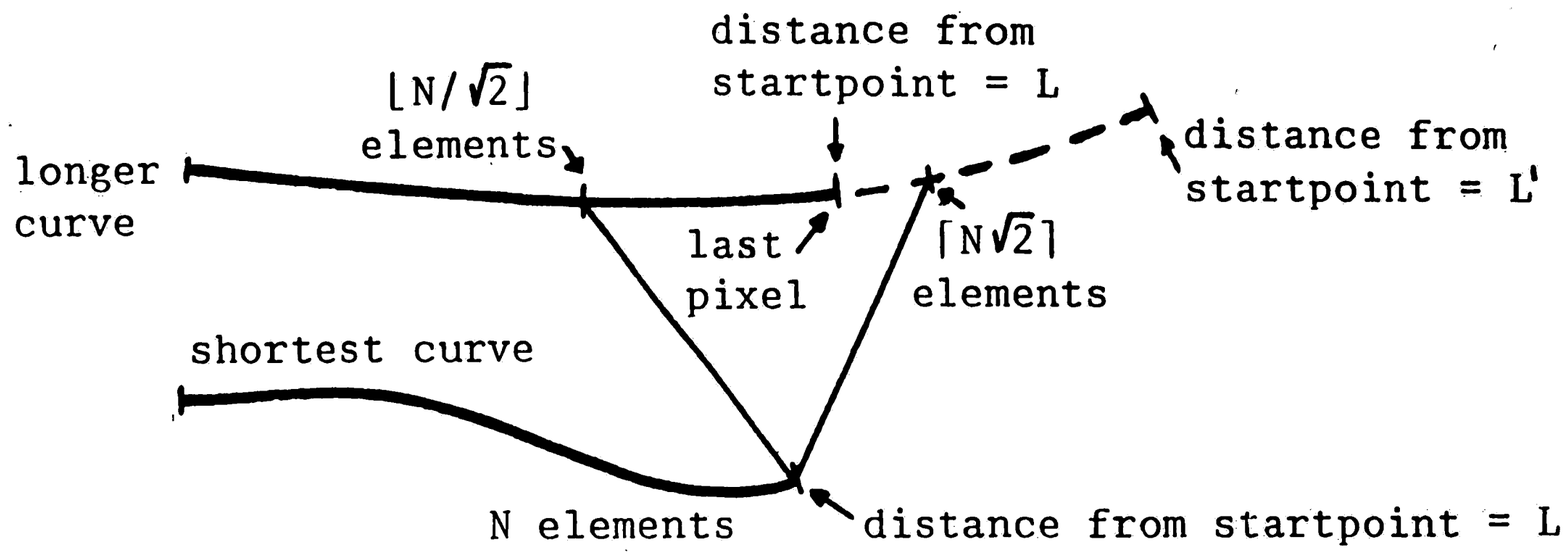


Figure 4.5 Location of last pixel to be included.

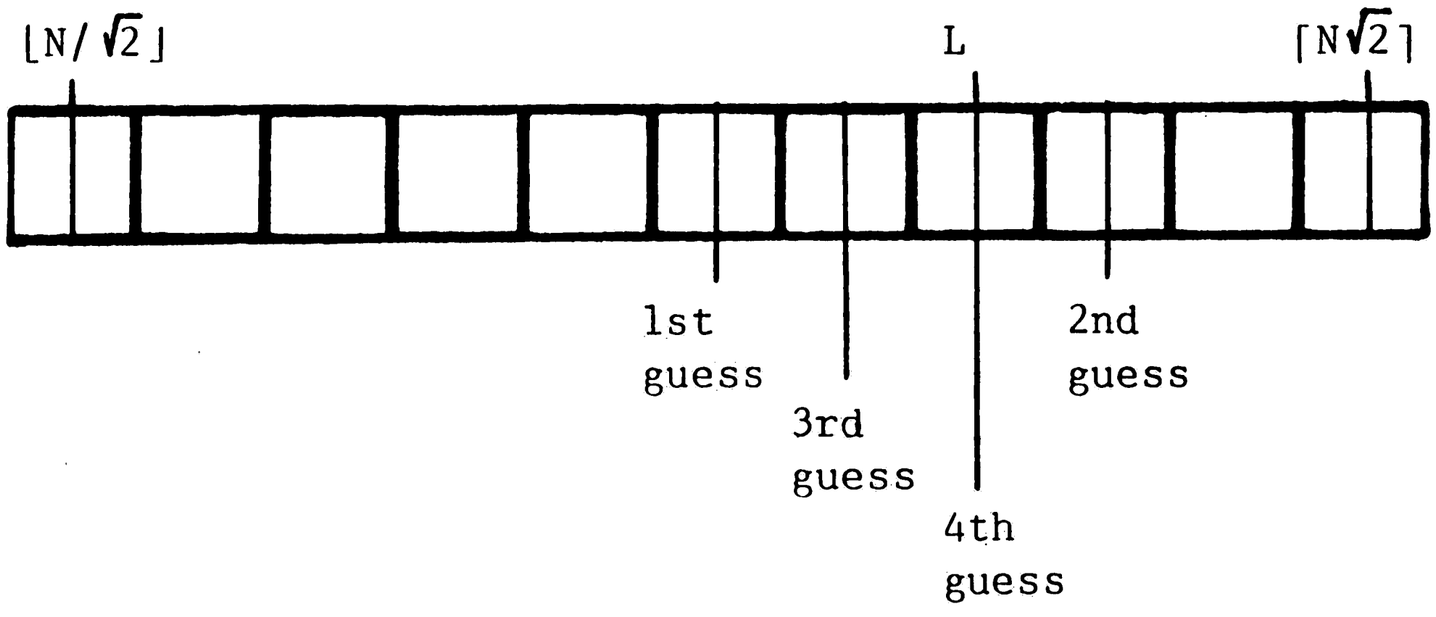


Figure 4.6 Binary search for exact pixel.

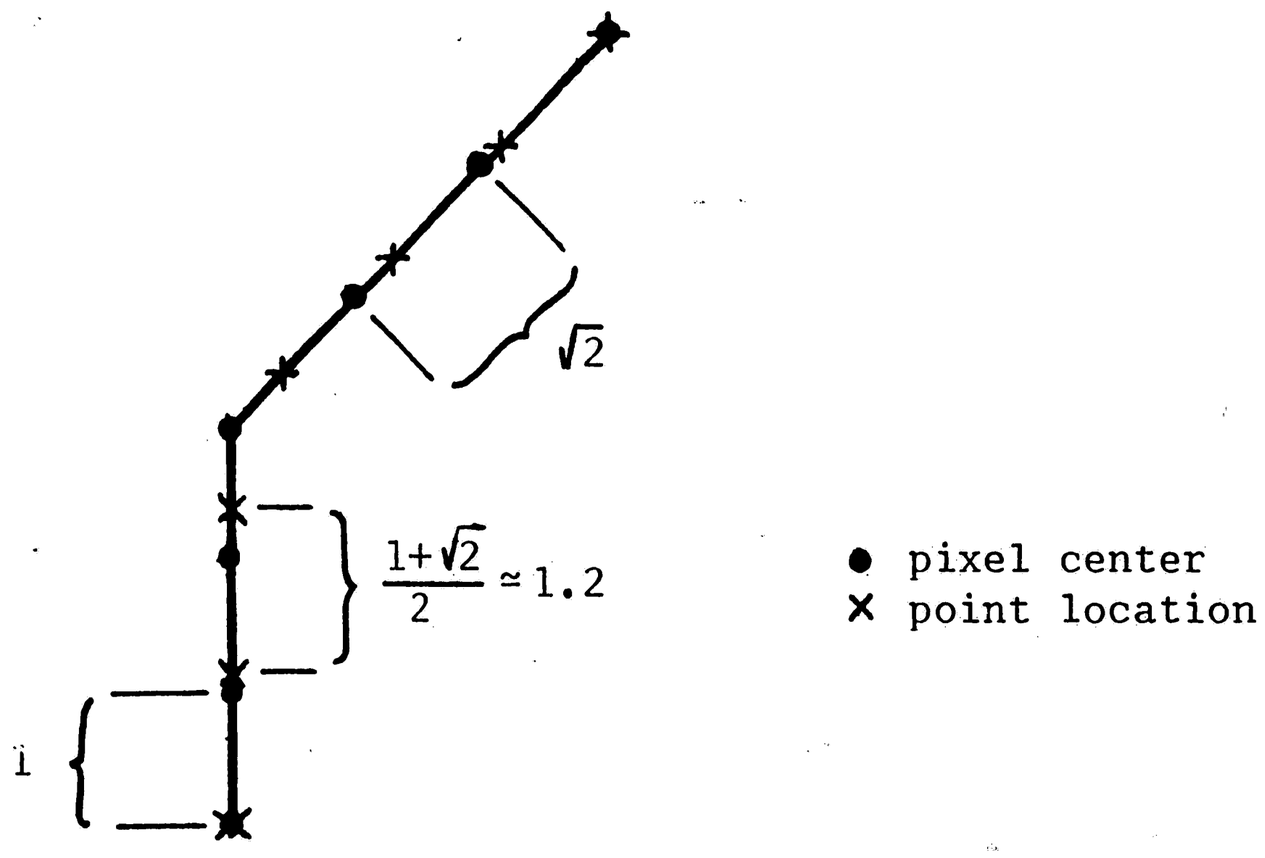


Figure 4.7 Location of points that will be averaged.

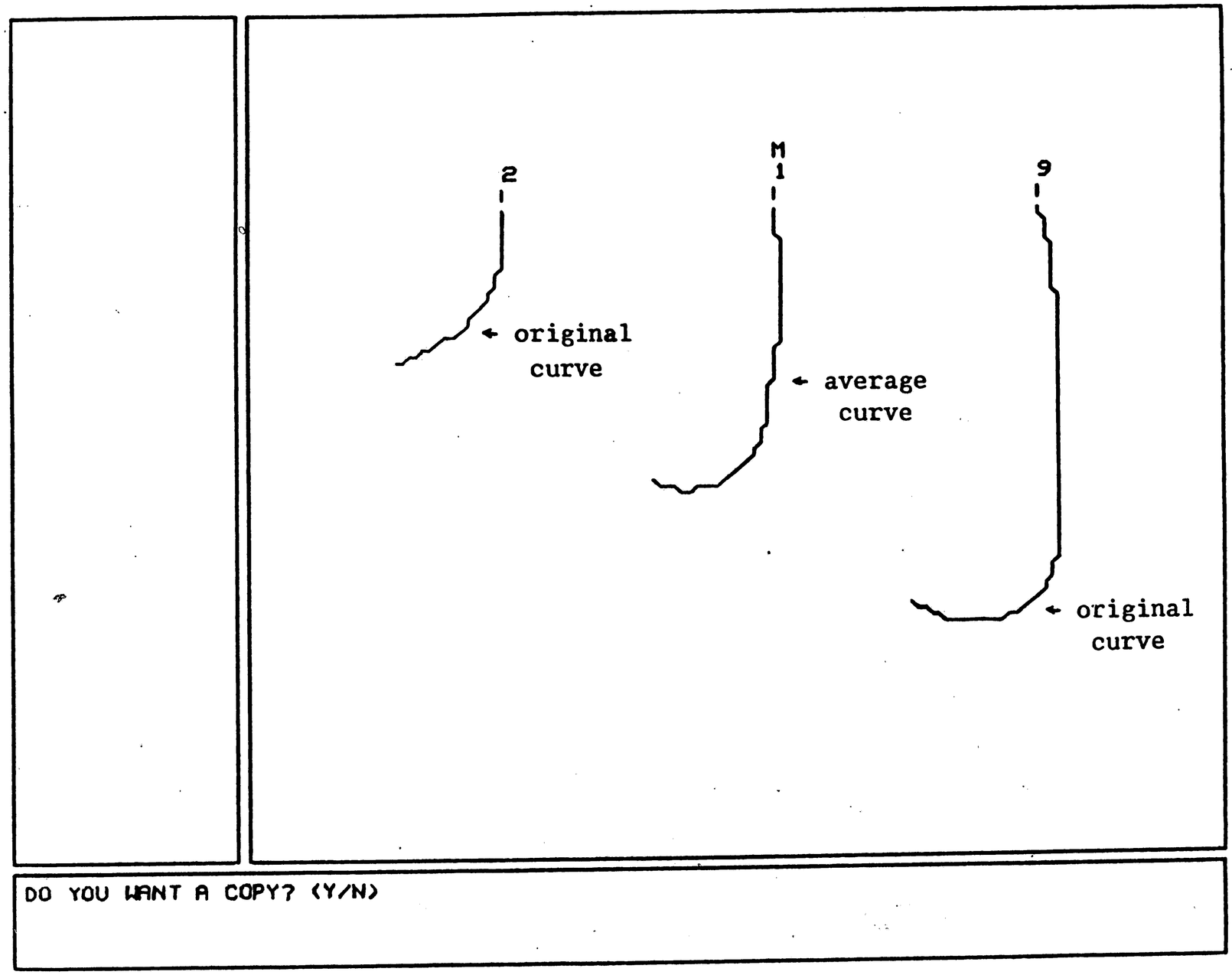


Figure 4.8 Unweighted average of two curves (using normalized lengths).

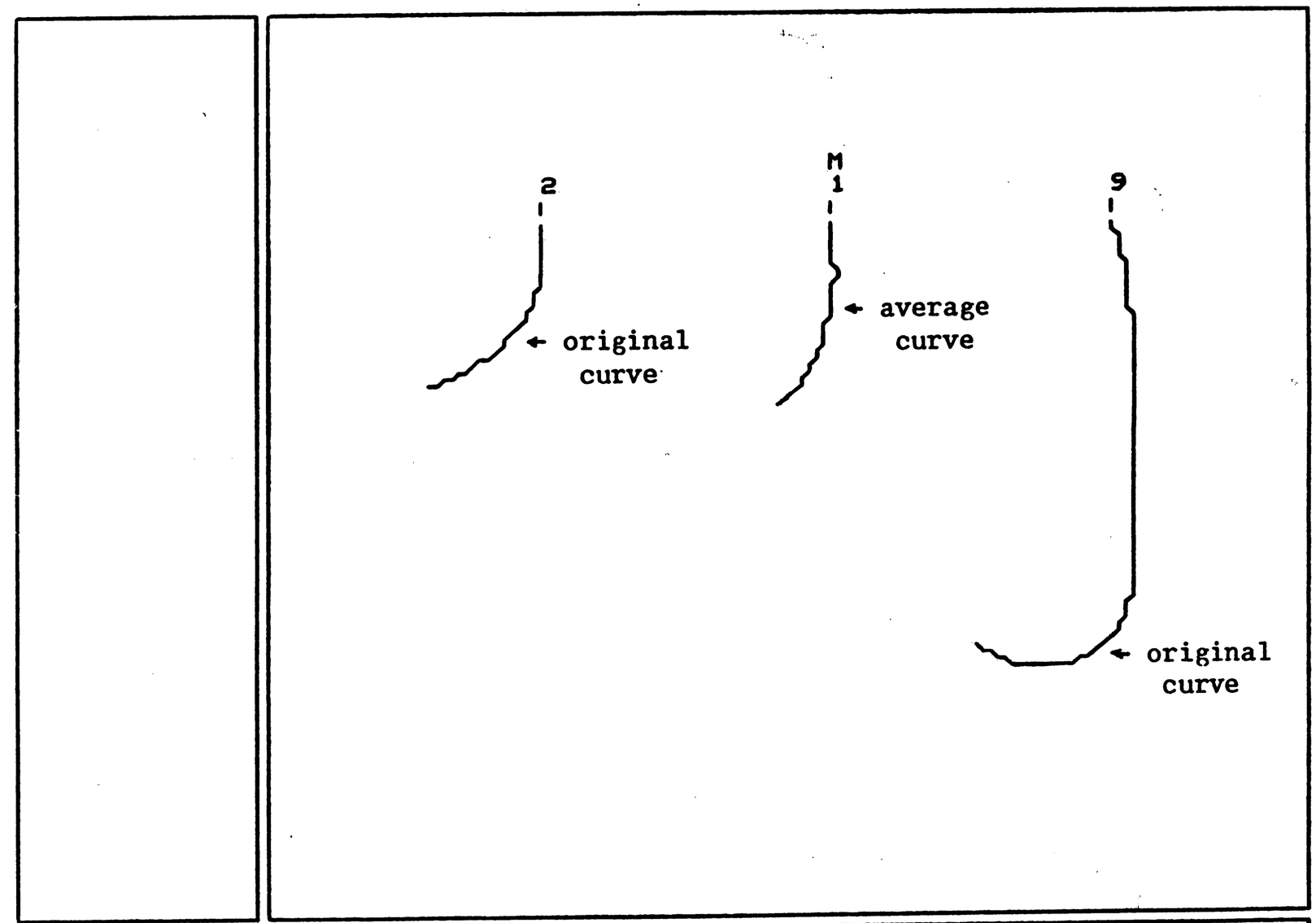


Figure 4.9 Unweigthed average of two curves (using shortest curve length).

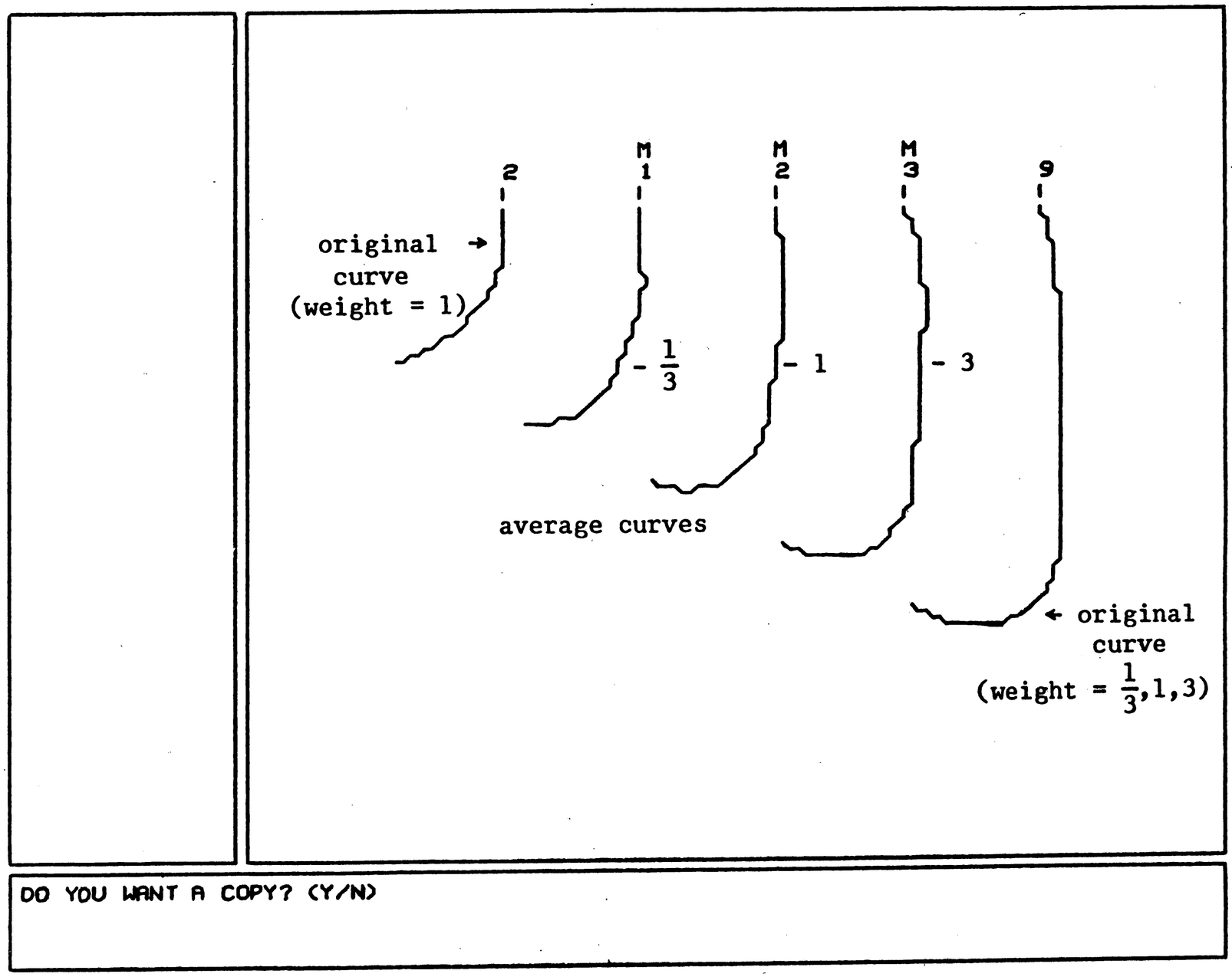


Figure 4.10 Weighted average of two curves (using normalized lengths).

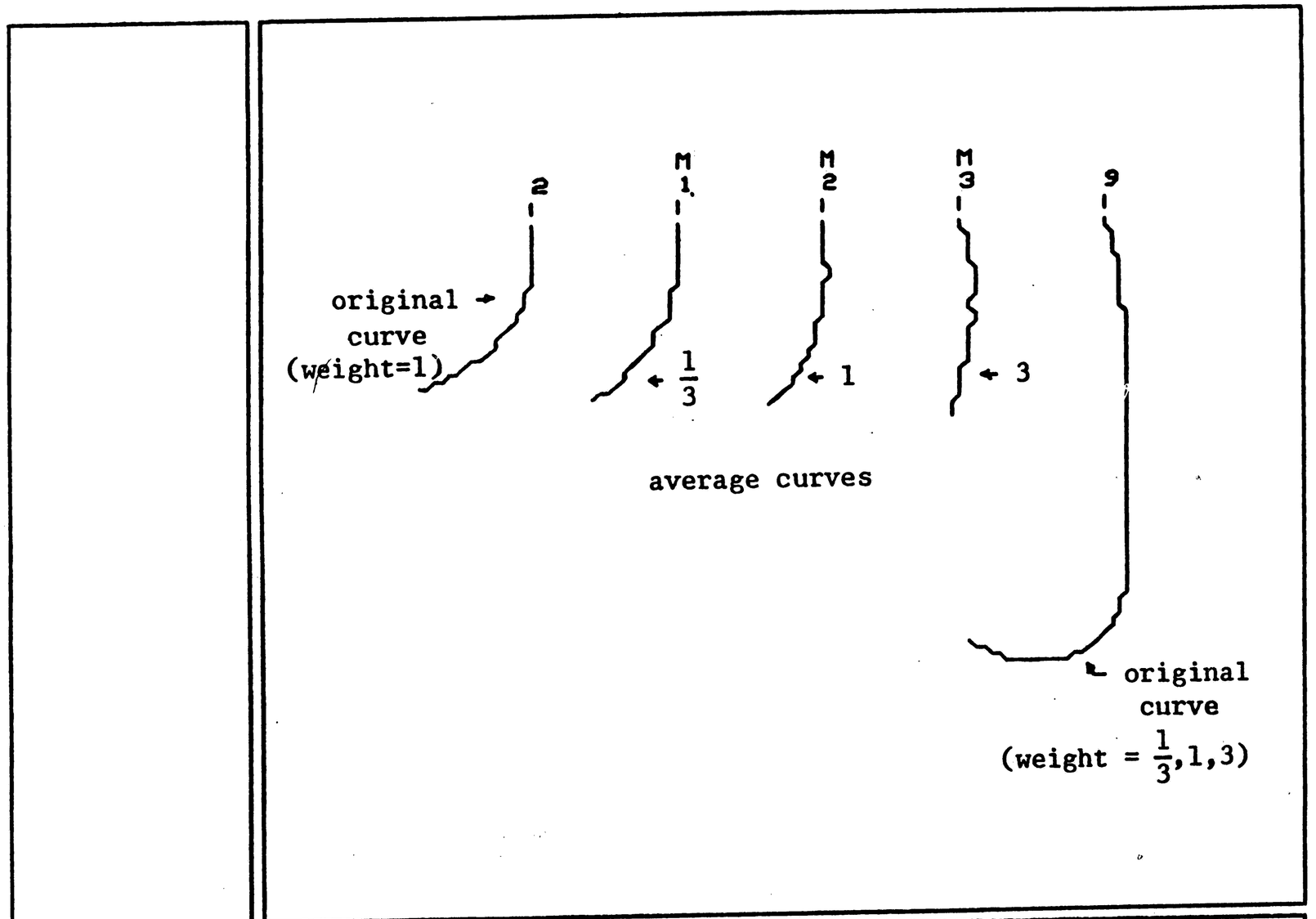


Figure 4.11 Weighted average of two curves (using shortest curve length).

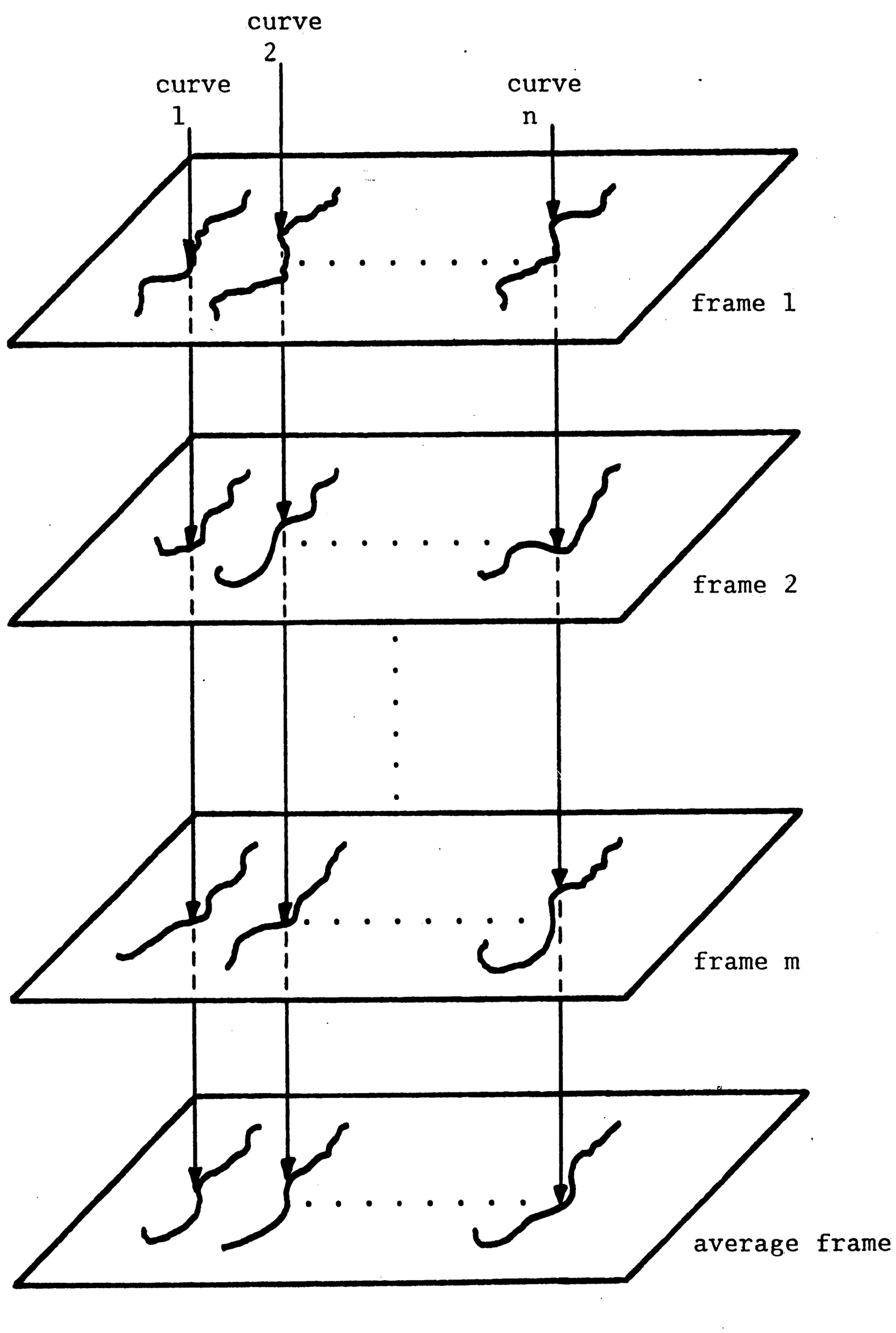


Figure 4.12 Averaging the curves of a set of frames.

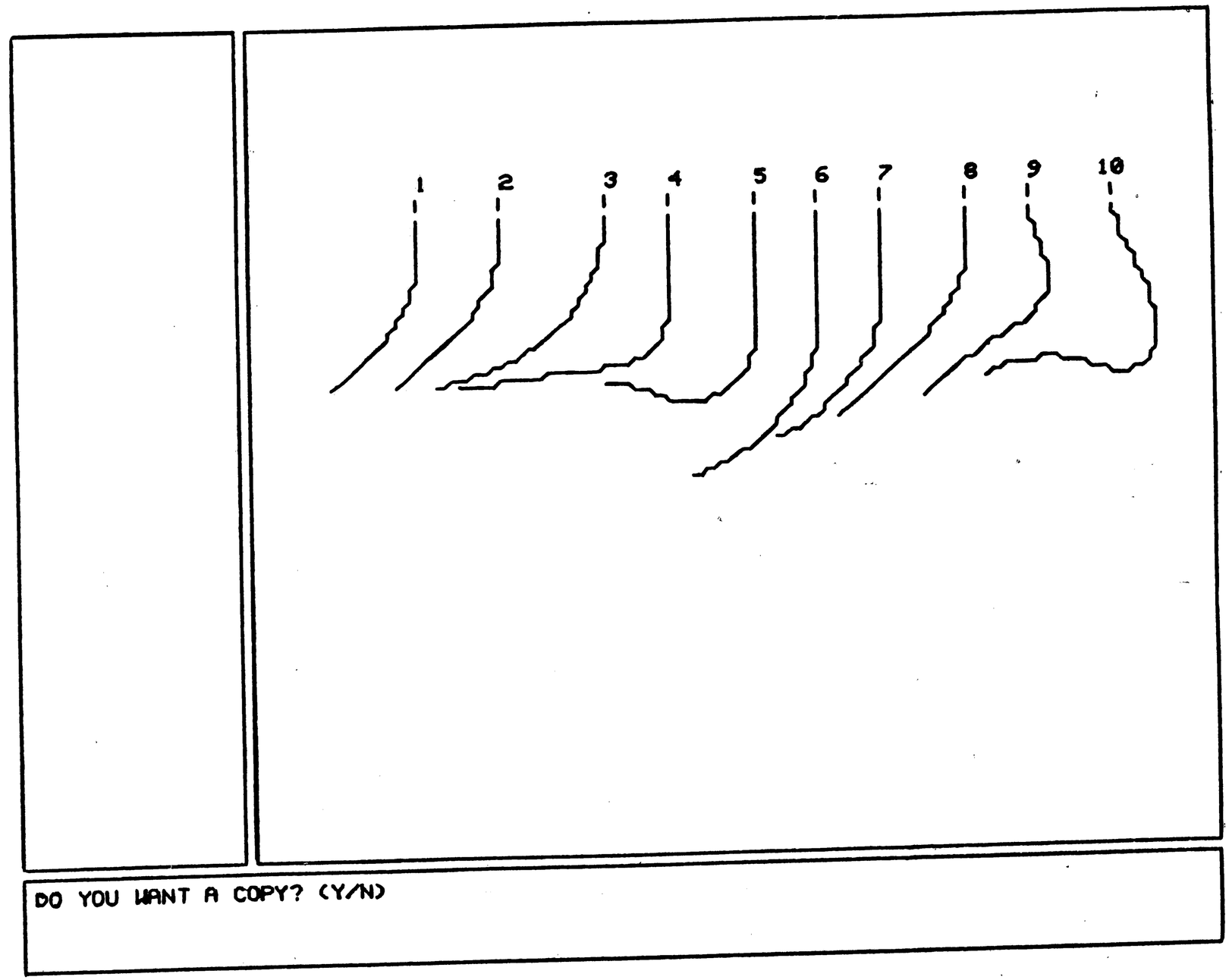


Figure 4.13 Top half of a fluid flow image.

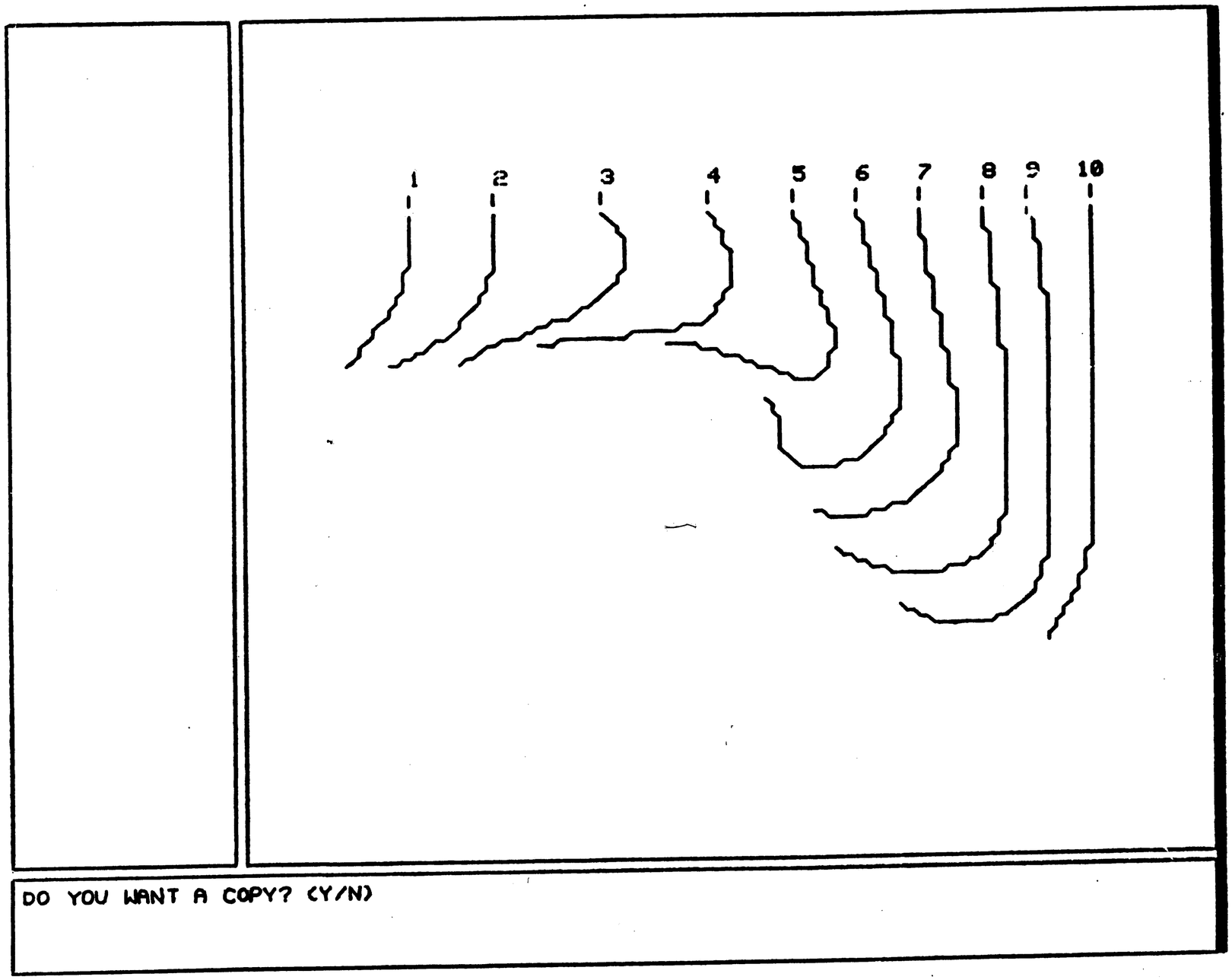


Figure 4.14 Top half of a fluid flow image (not equal to Figure 4.13).

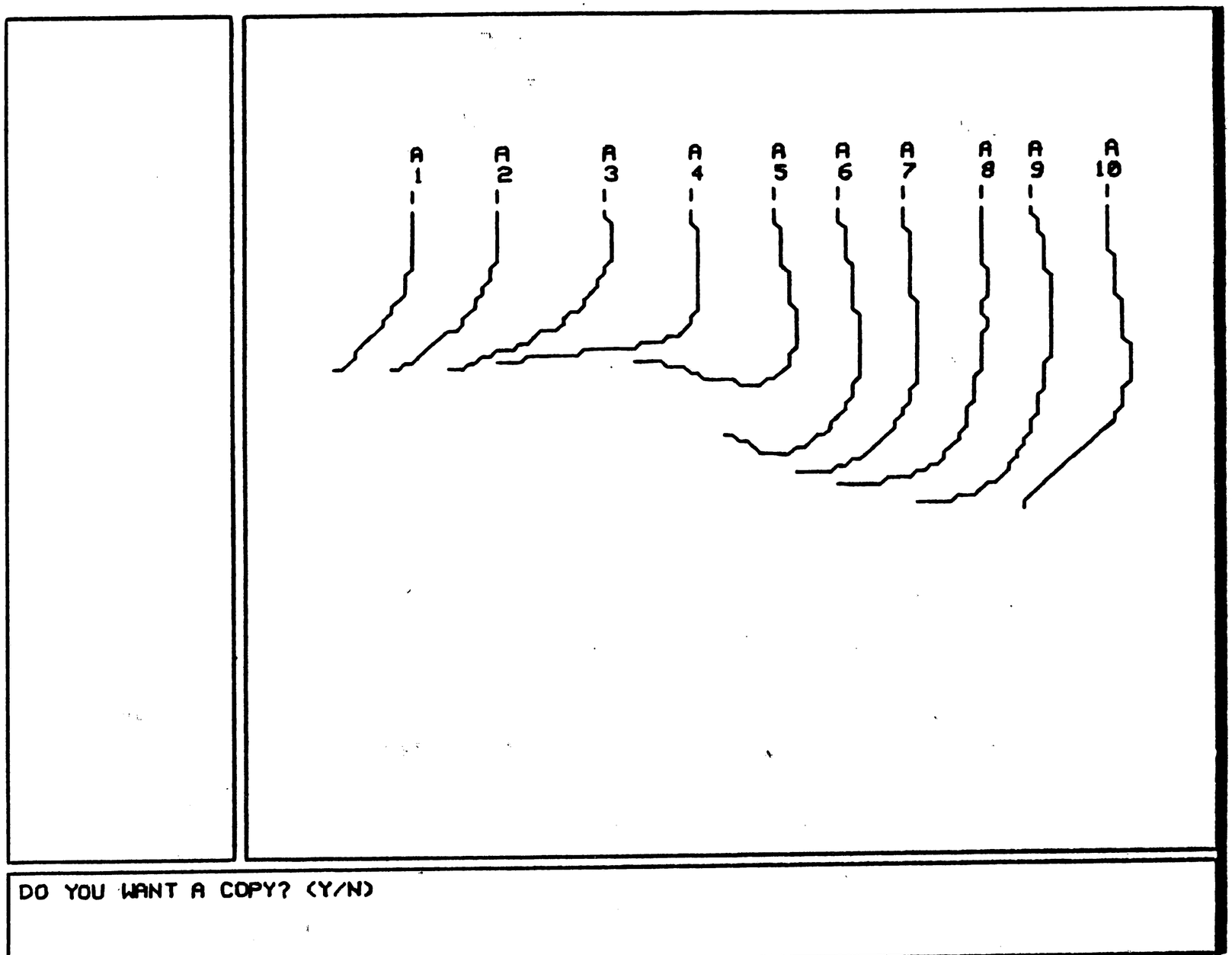


Figure 4.15 Top half of average frame (using normalized lengths).

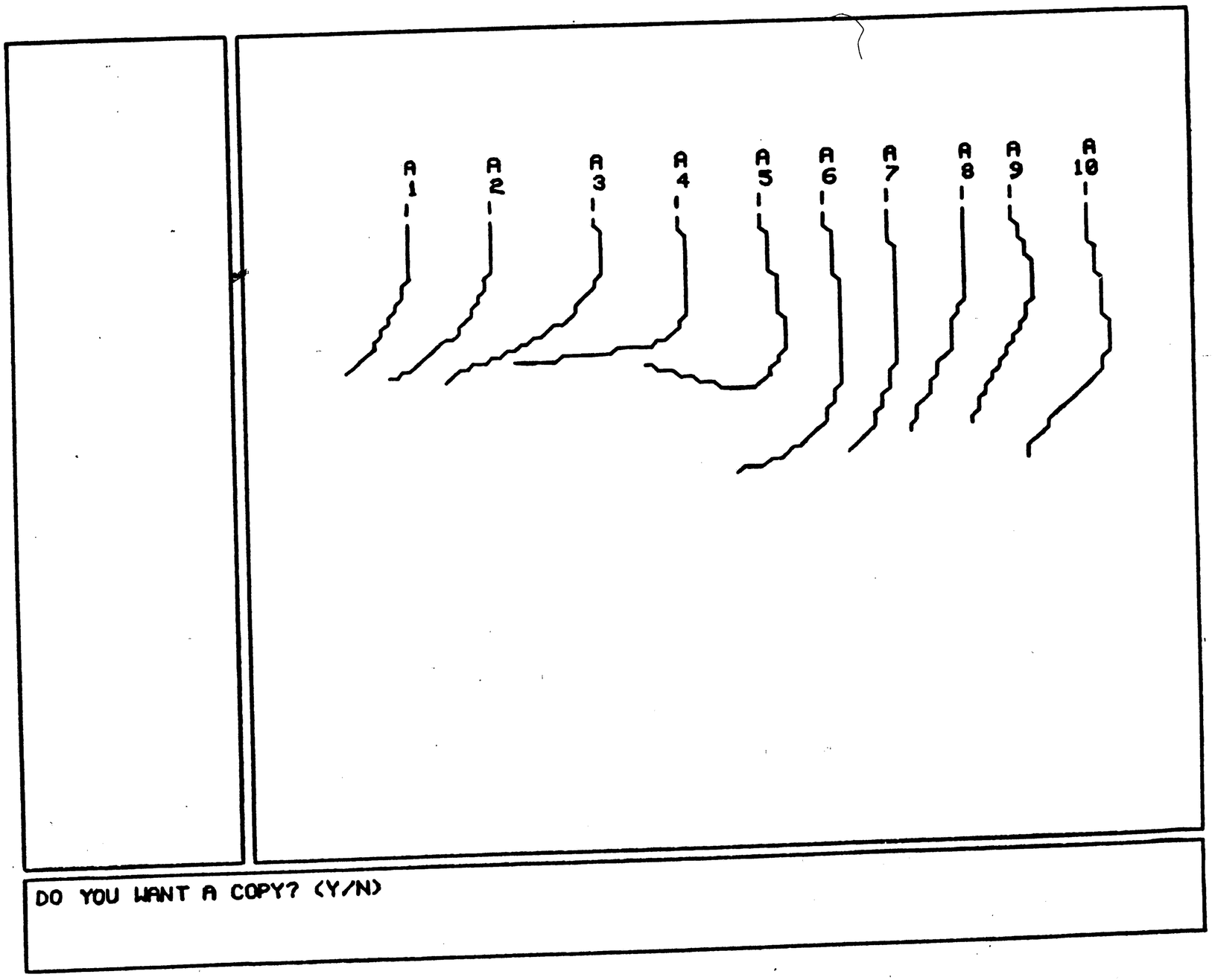


Figure 4.16 Top half of average frame (using shortest curve length).

element #	1	2	3	4	5	6	7	8	absolute displacement	average displacement
Δx	1	1	0	-1	0	-1	-1	-1	-2	$-2/8 = -\frac{1}{4}$
Δy	0	1	1	1	1	1	0	-1	+4	$4/8 = \frac{1}{2}$
cumulative average Δx	$-\frac{1}{4}$	$-\frac{1}{2}$	$-\frac{3}{4}$	-1	$-1\frac{1}{4}$	$-1\frac{1}{2}$	$-1\frac{3}{4}$	-2	-2	$-\frac{1}{4}$
cumulative average Δy	$\frac{1}{2}$	1	$1\frac{1}{2}$	2	$2\frac{1}{2}$	3	$3\frac{1}{2}$	4	+4	$\frac{1}{2}$
rounded cumulative Δx	0	0	-1	-1	-1	-1	-2	-2	-2	$-\frac{1}{4}$
rounded cumulative Δy	0	1	1	2	2	3	3	4	4	$\frac{1}{2}$
smoothed Δx	0	0	-1	0	0	0	-1	0	-2	$-\frac{1}{4}$
smoothed Δy	0	1	0	1	0	1	0	1	4	$\frac{1}{2}$

Figure 5.1 Δx and Δy sequences a particular curve.

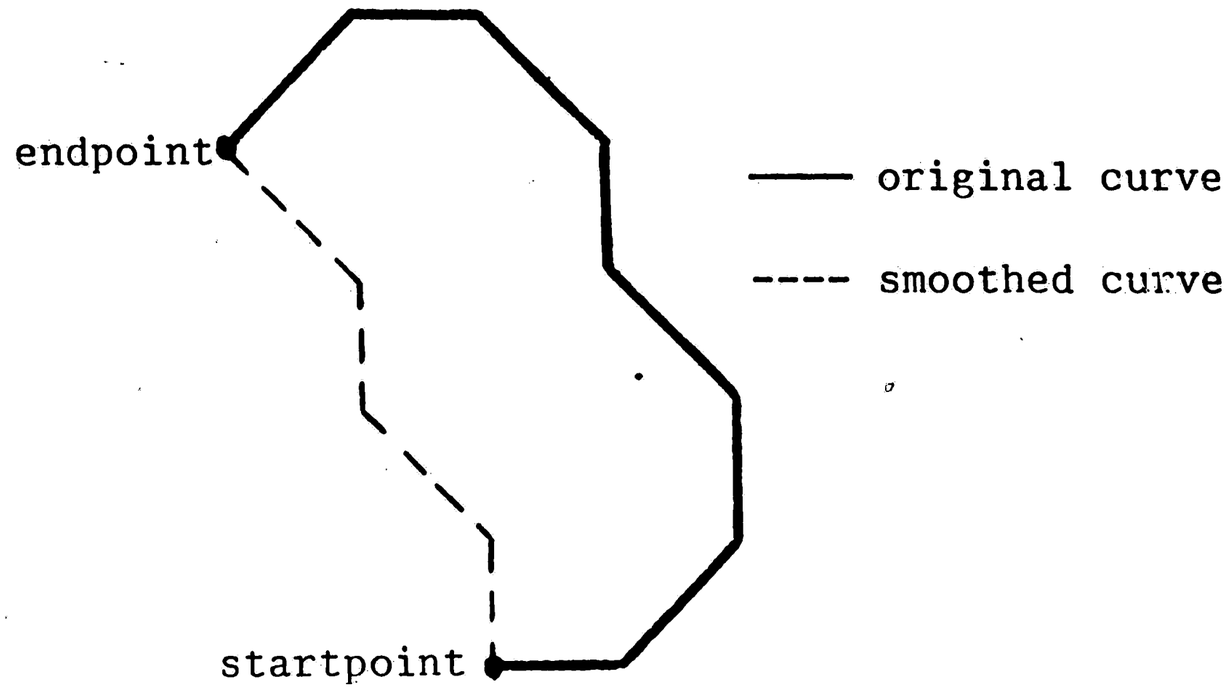


Figure 5.2 Original curve and smoothed curve.

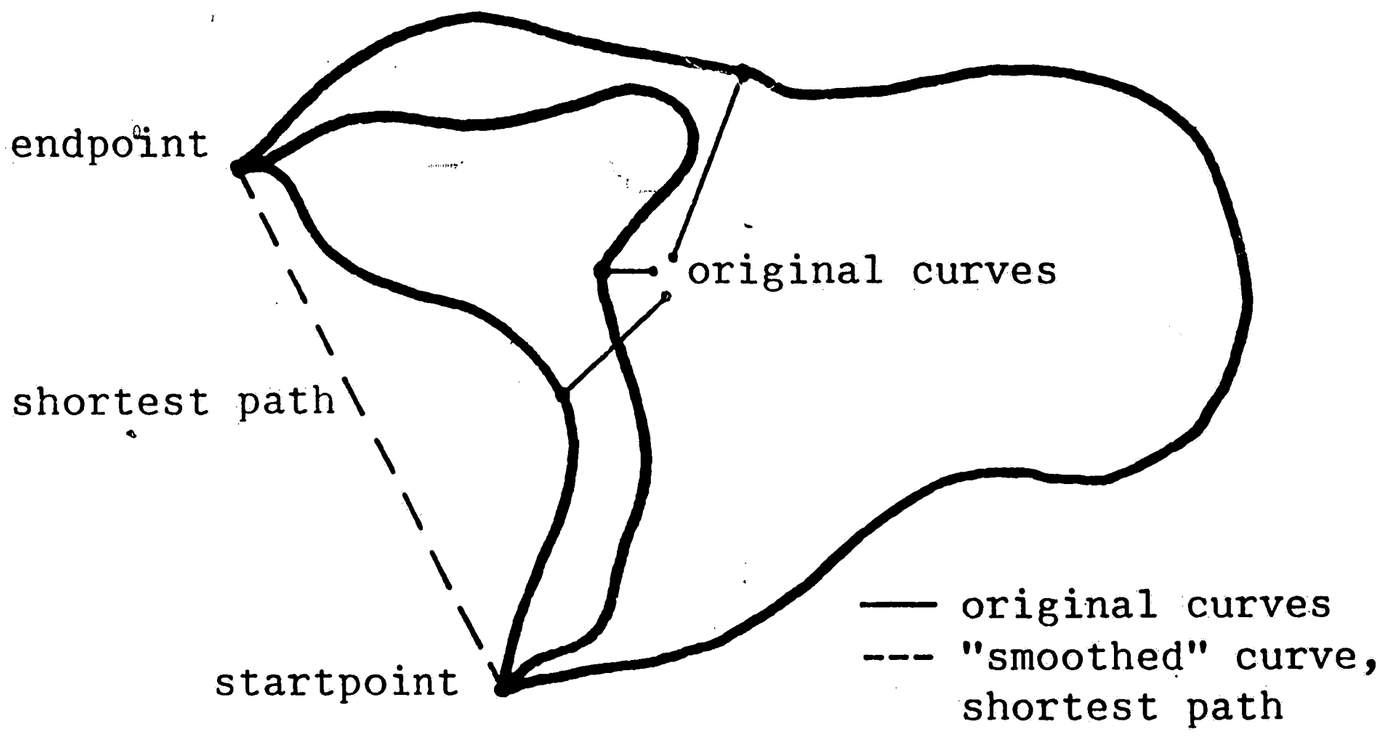


Figure 5.3 Shortest path between startpoint and endpoint of original curves.

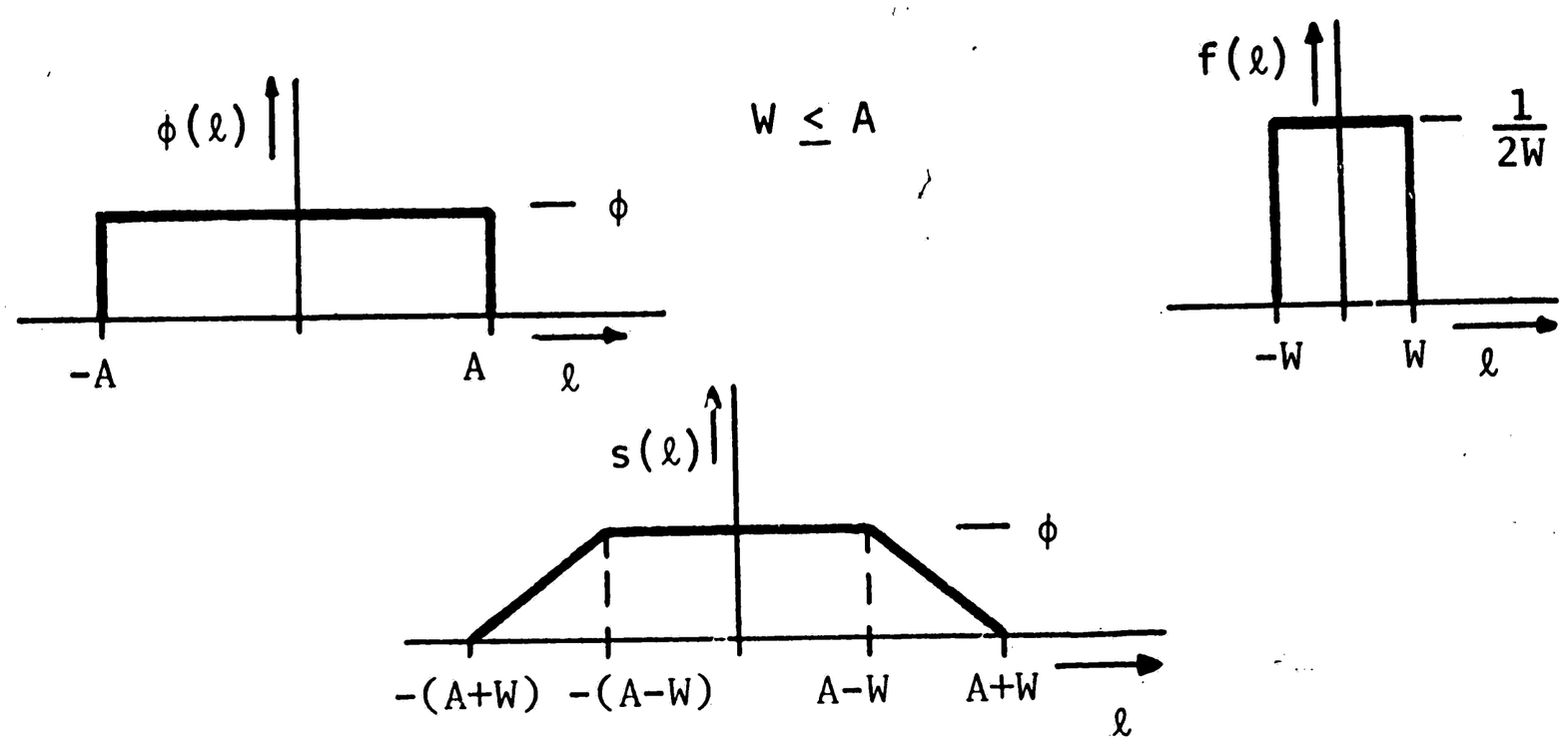


Figure 5.4 Convolution operation.

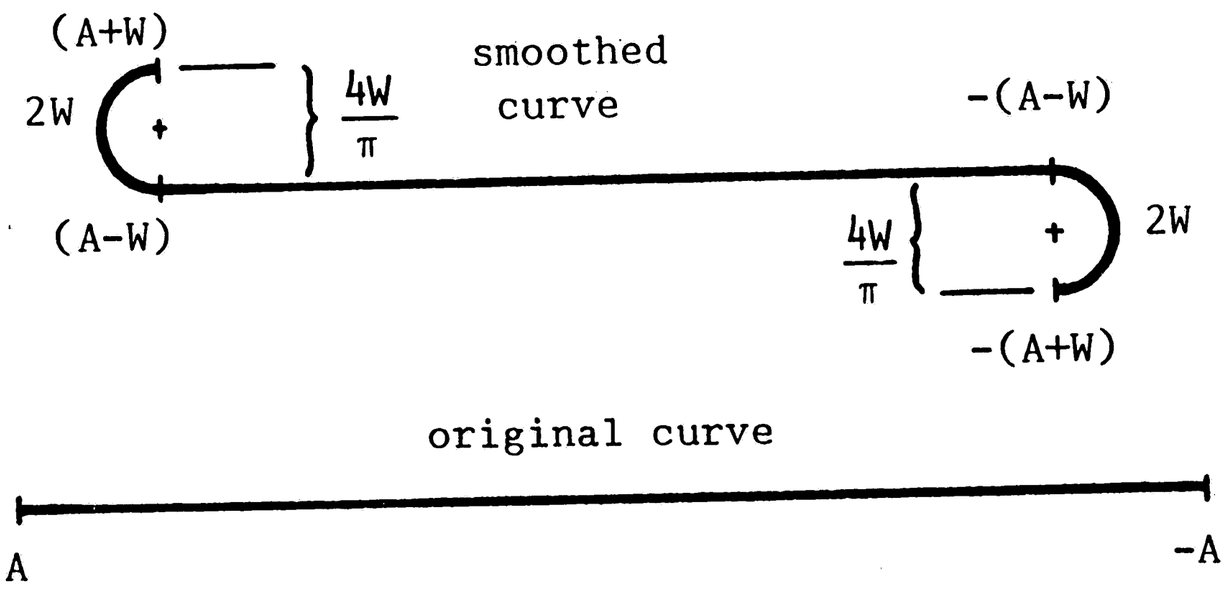


Figure 5.5 Smoothed curve with $\phi=\pi$.

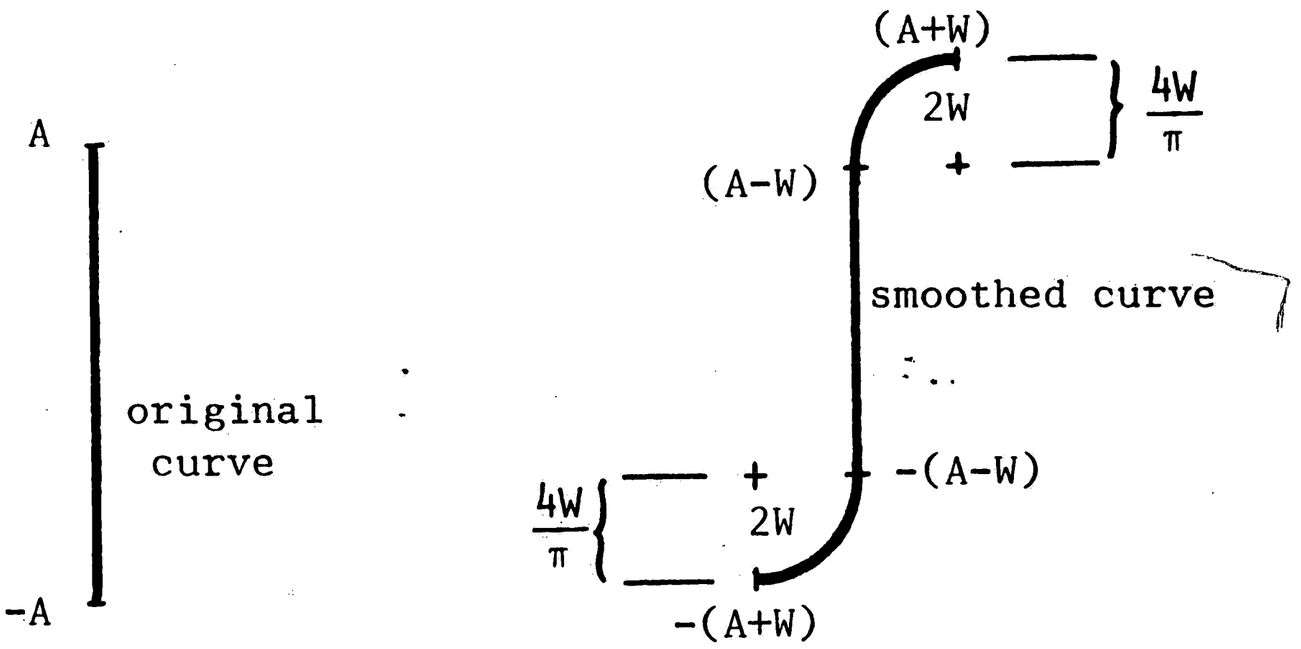
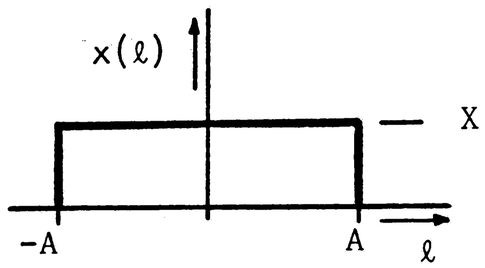
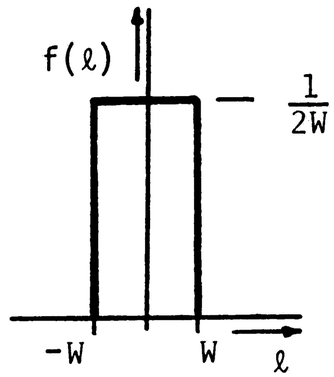
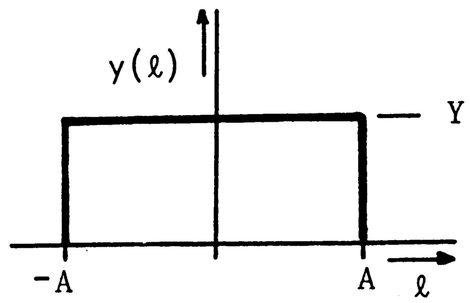


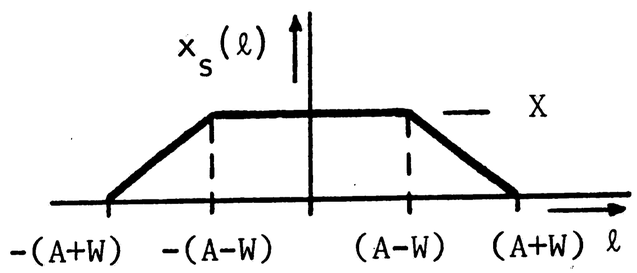
Figure 5.6 Smoothed curve with $\phi=\frac{\pi}{2}$.







$$\phi = \arctan Y/X$$



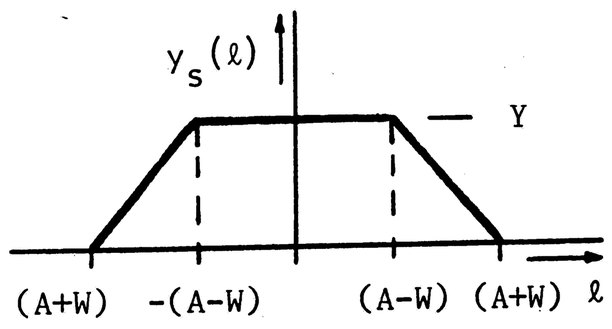
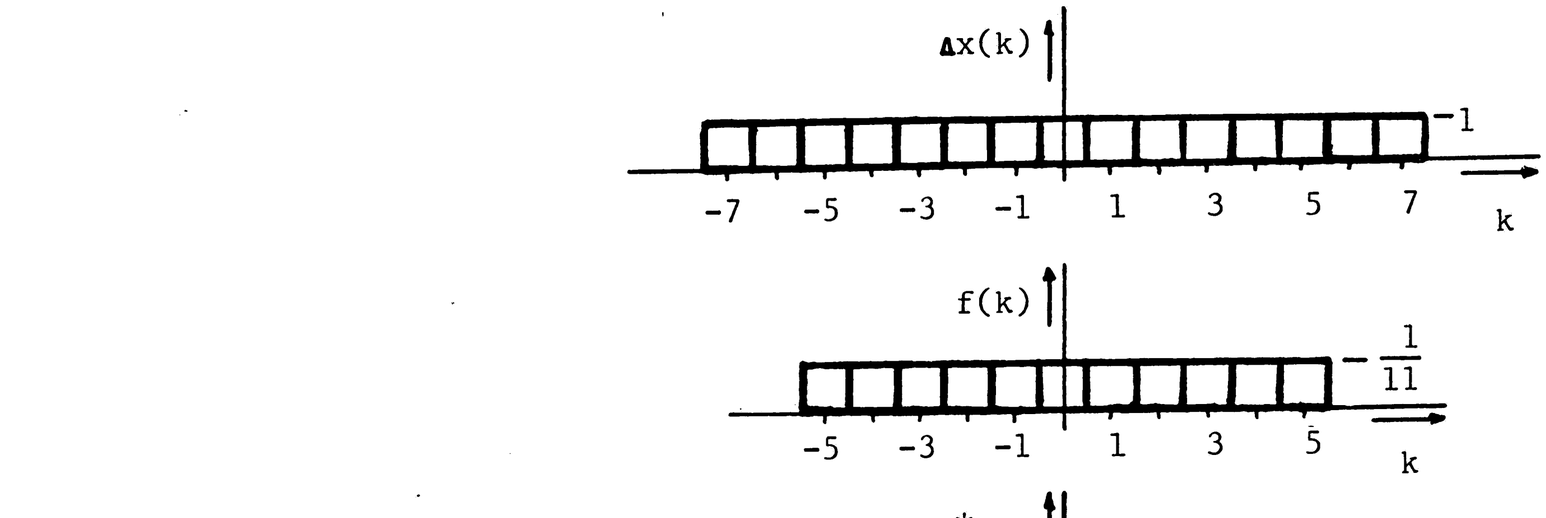


Figure 5.7 Smoothed x and y coordinate functions.



cumulative error

error →

0	-1/11	-3/11	5/11	1/11	-4/11	1/11	5/11	-3/11	-1/11	0	0	0	0	0	-1/11	-3/11	5/11	1/11	-4/11	1/11	5/11	-3/11	1/11	0
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12

k

Figure 5.8 Discrete smoothed curve.

k	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
Δx				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
Δx_s^*	1			1	1		1	1	1	1	1	1	1	1	1		1	1			1
Δy				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
Δy_s^*	1			1	1		1	1	1	1	1	1	1	1	1		1	1			1

Figure 5.9 Original and smoothed curve data.

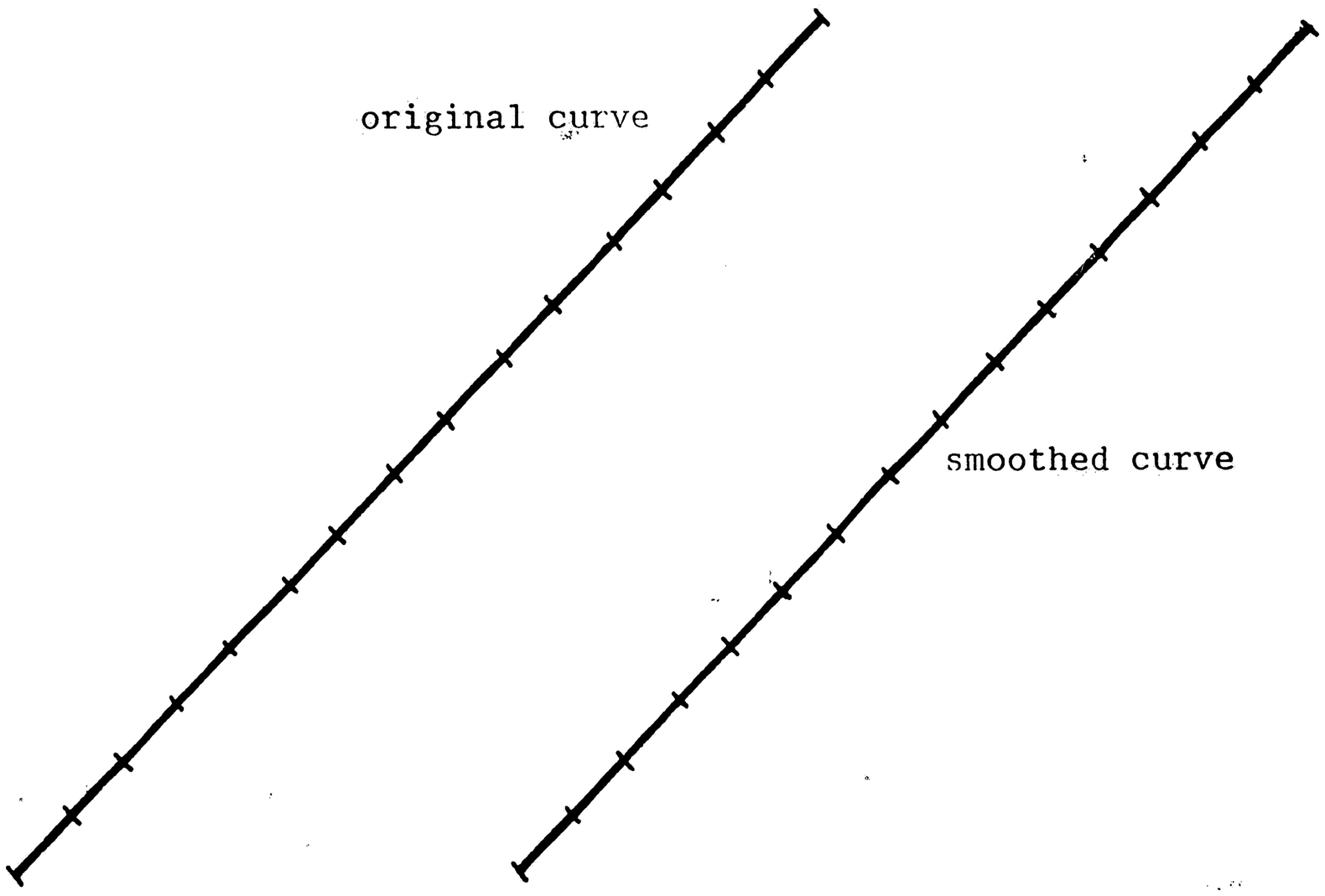


Figure 5.10 Original and smoothed curves from data of Figure 5.9.

k	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10
Δx				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
Δx_s^*	1			1	1		1	1	1	1	1	1	1	1	1		1	1			1
Δy				0	1	1	1	1	1	1	1	1	1	1	1	1	1	0			
Δy_s^*		1			1	1		1	1	1	1	1	1	1		1	1			1	

Figure 5.11 original and smoothed curve data.

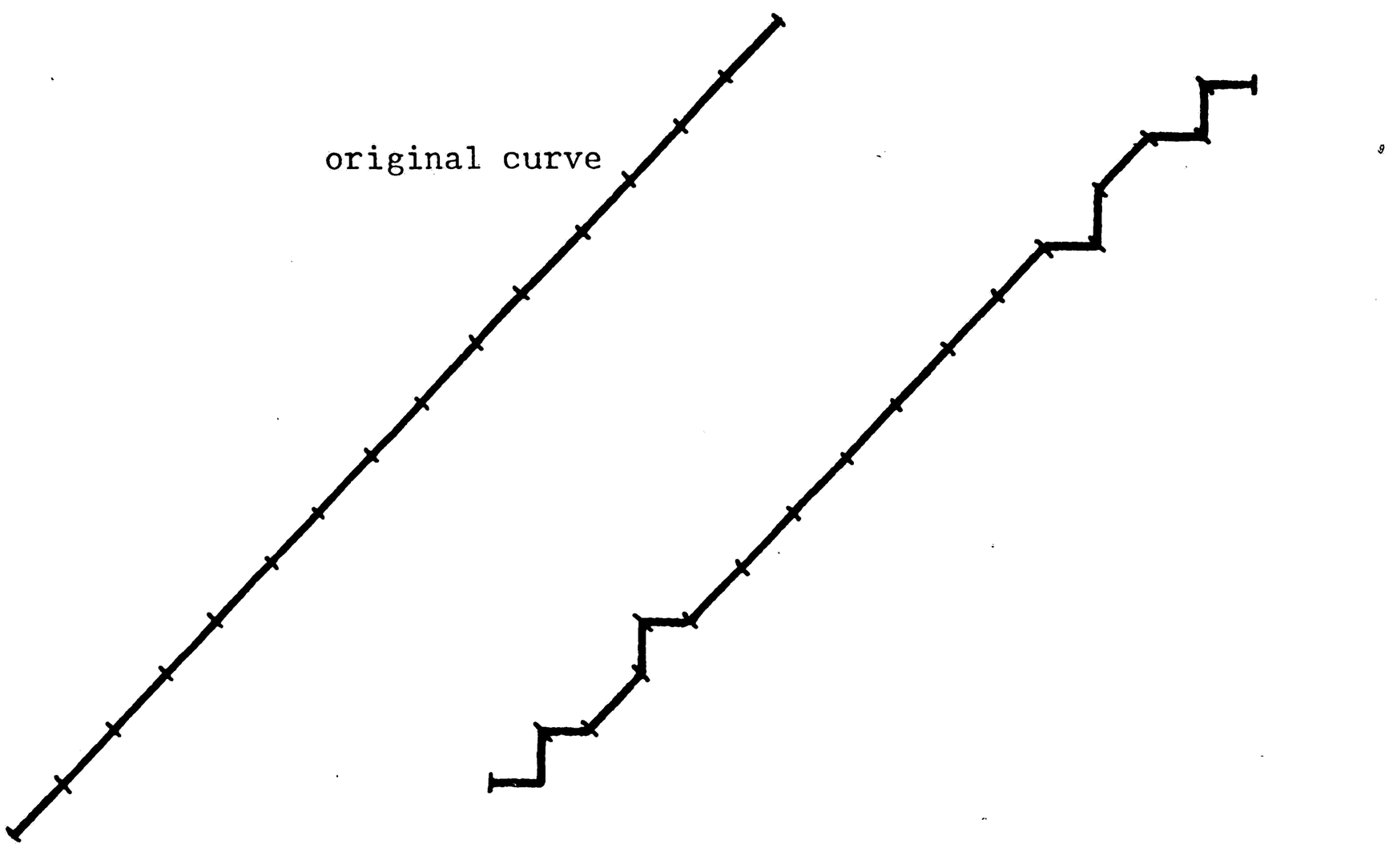


Figure 5.12 Original and smoothed curves from Figure 5.11.

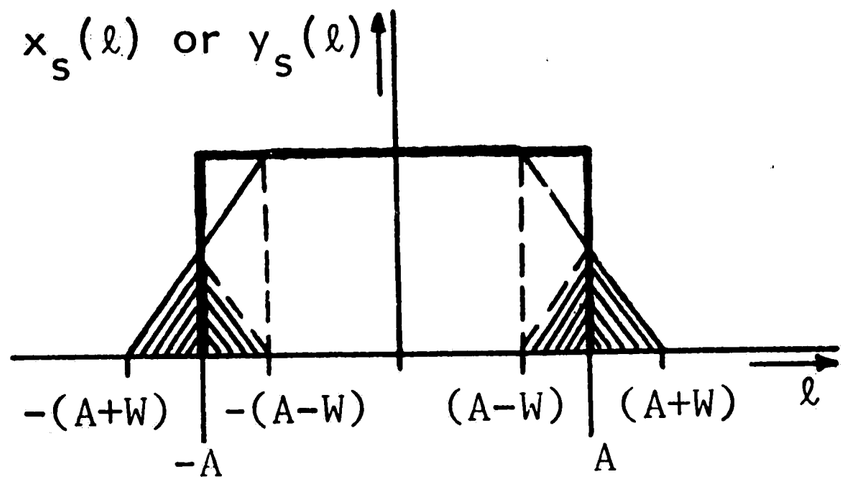


Figure 5.13 Modified smoothed tangent angle function.

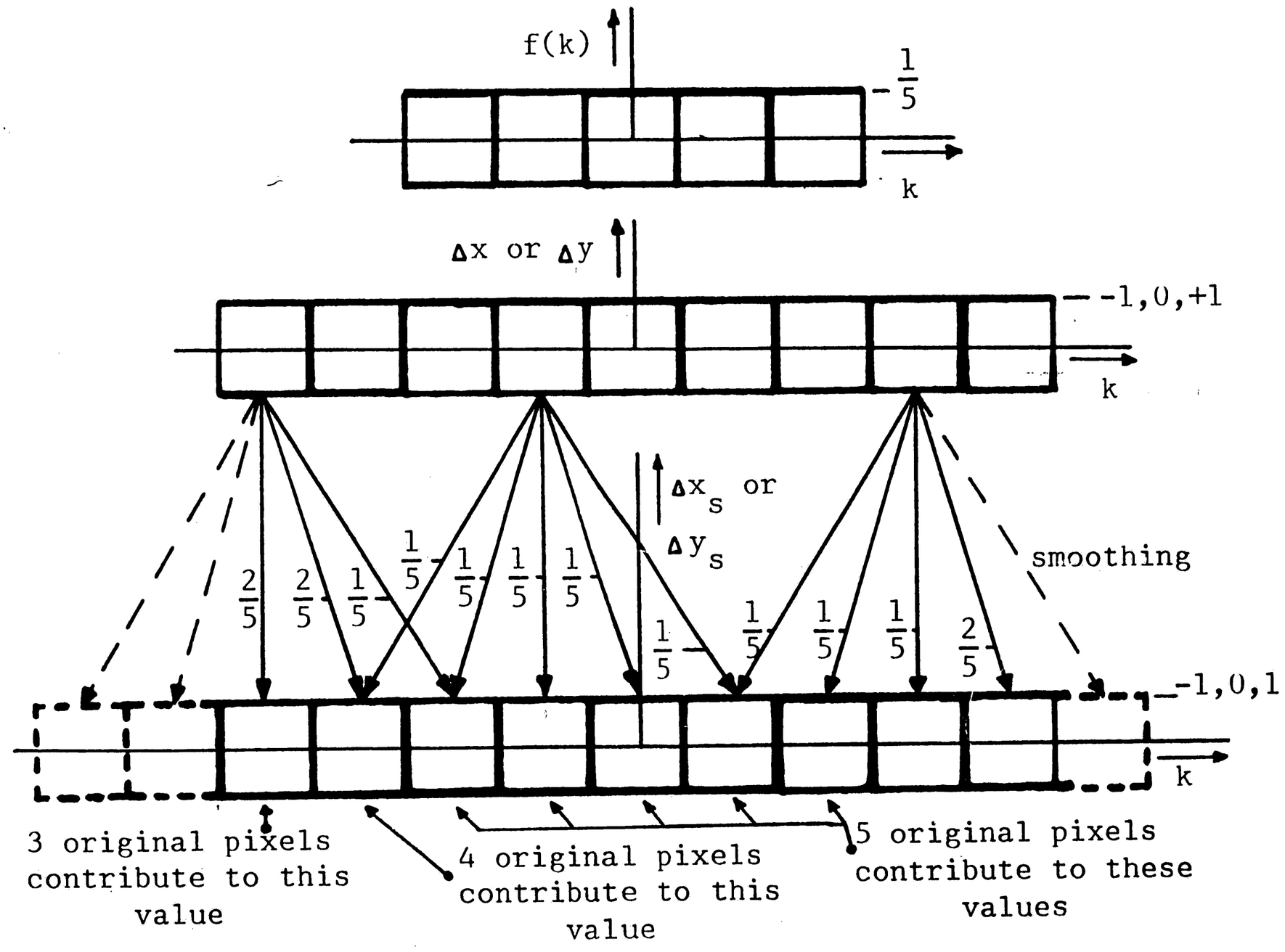


Figure 5.14 Weighting of displacement values.

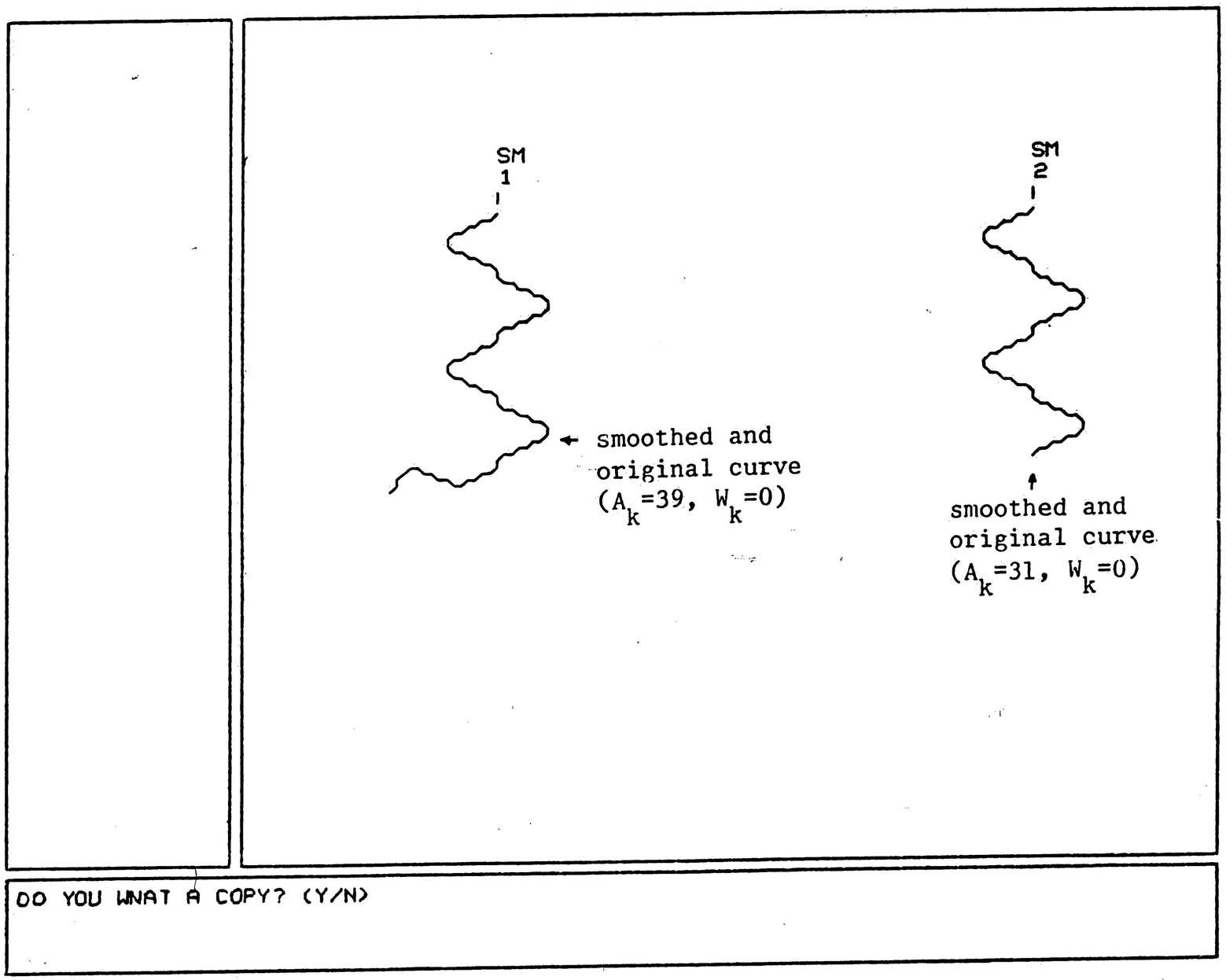


Figure 5.15 Smoothed and original curves (W_k = 0).

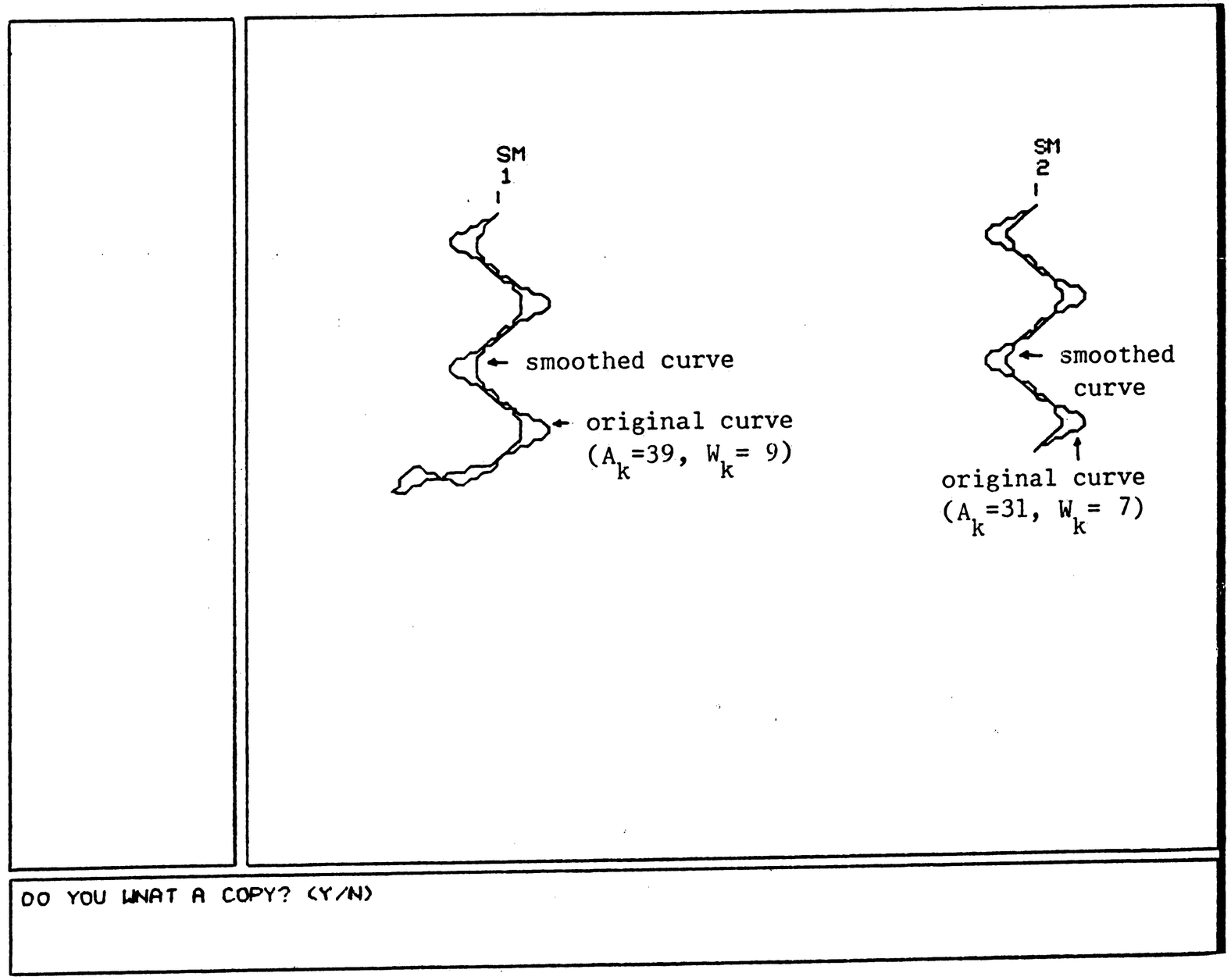


Figure 5.16 Smoothed and original curves (same as in Figure 5.15; $W_k \simeq 0.25 \times A_k$).

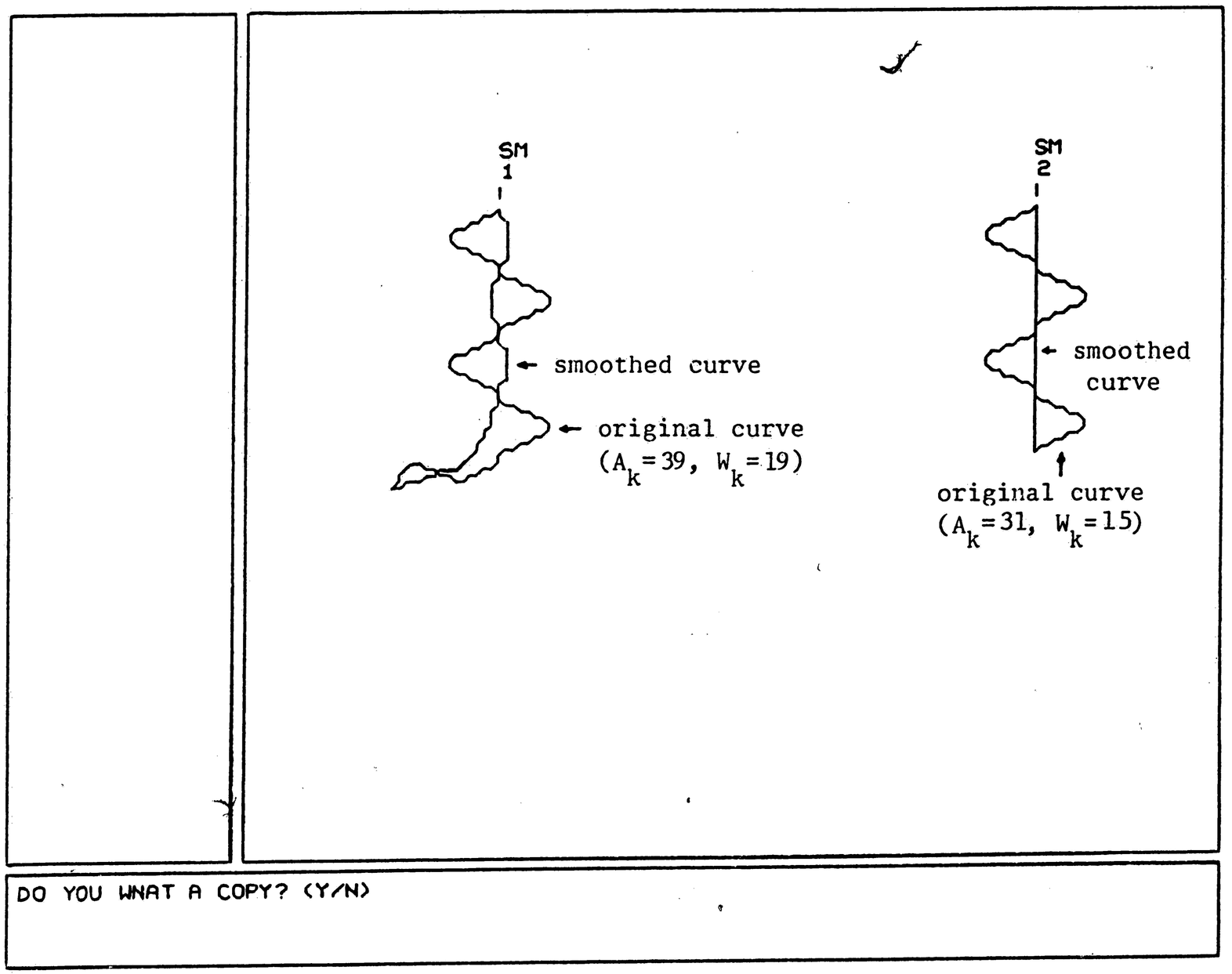


Figure 5.17. Smoothed and original curves (same as in Figure 5.15; $W_k \simeq 0.5 \times A_k$).

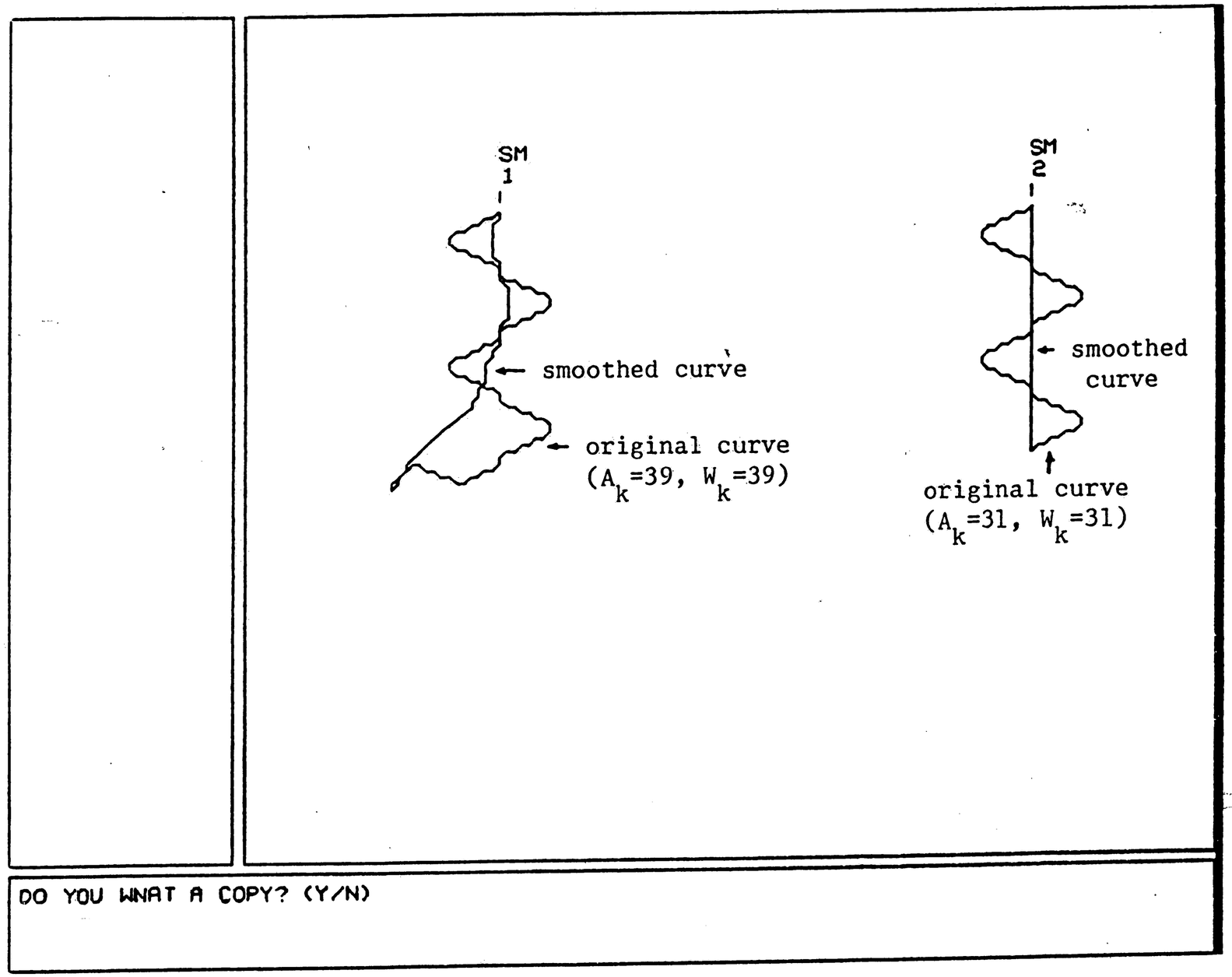


Figure 5.18 Smoothed and original curves (same as in Figure 5.15; $W_k = A_k$).

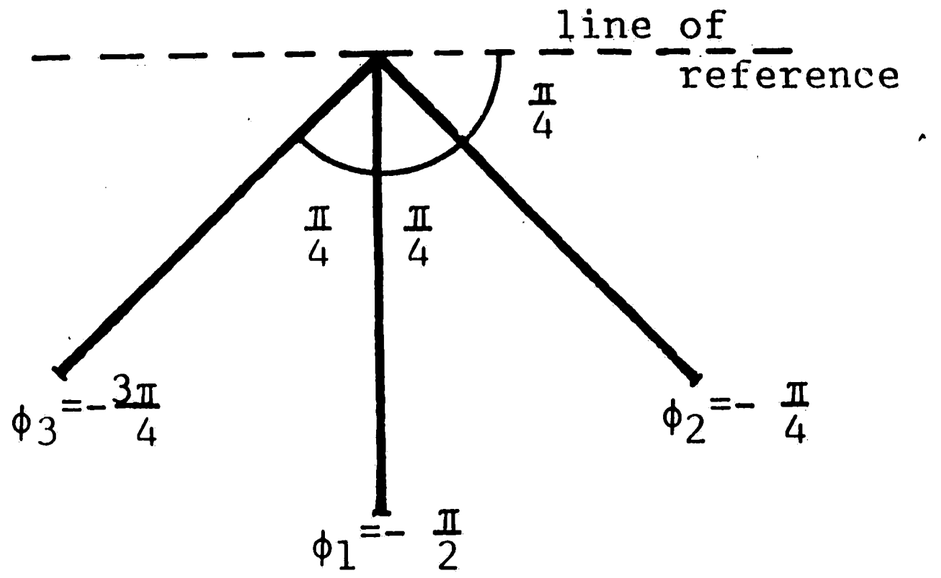


Figure 5.19 Three different curves all with length 2A.

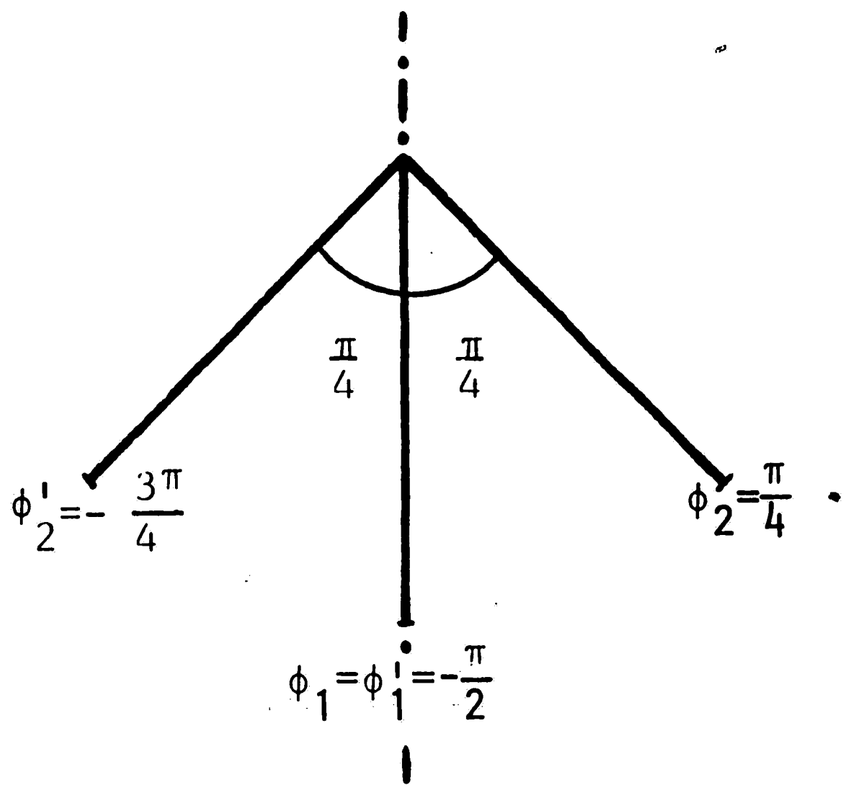


Figure 5.20 Mirrored and original curves $\phi 1$ and $\phi 2$.

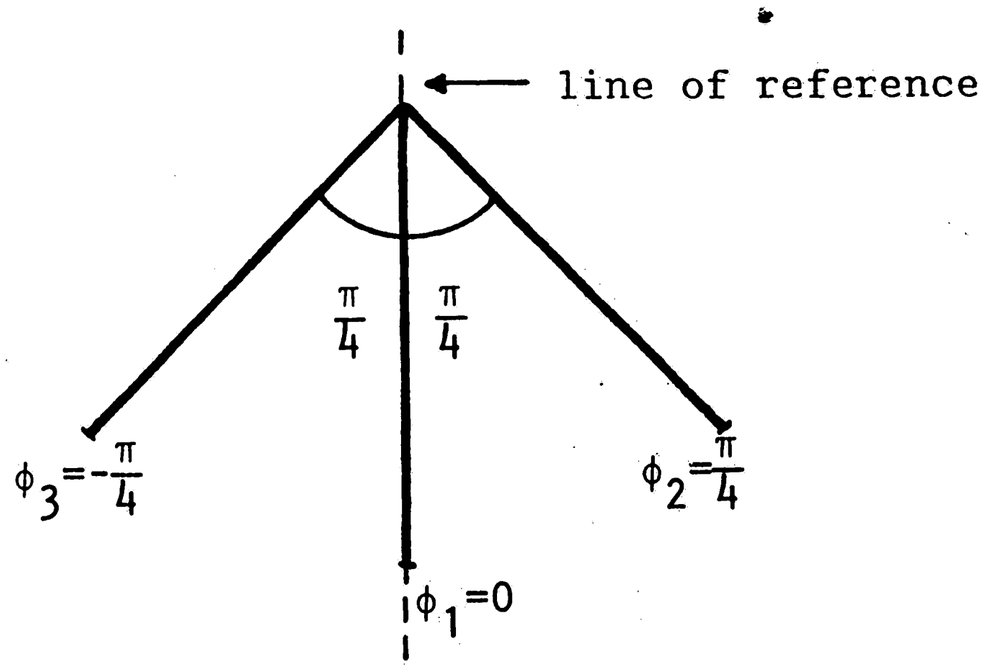


Figure 5.21 Curves of Figure 5.19 with different line of reference.

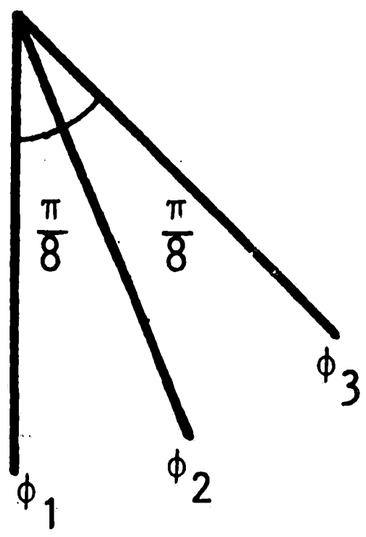


Figure 5.22 3 different curves all with length 2A.

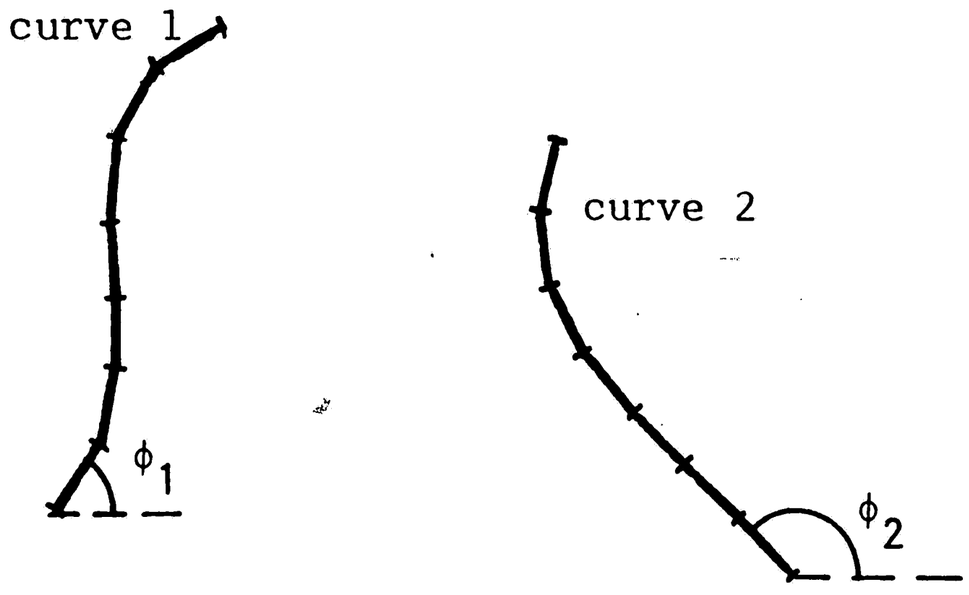


Figure 5.23 Curve elements to be compared (7 elements here).

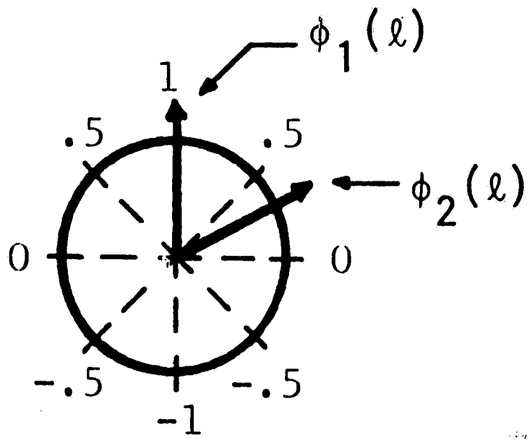


Figure 5.24 Element similarity function.

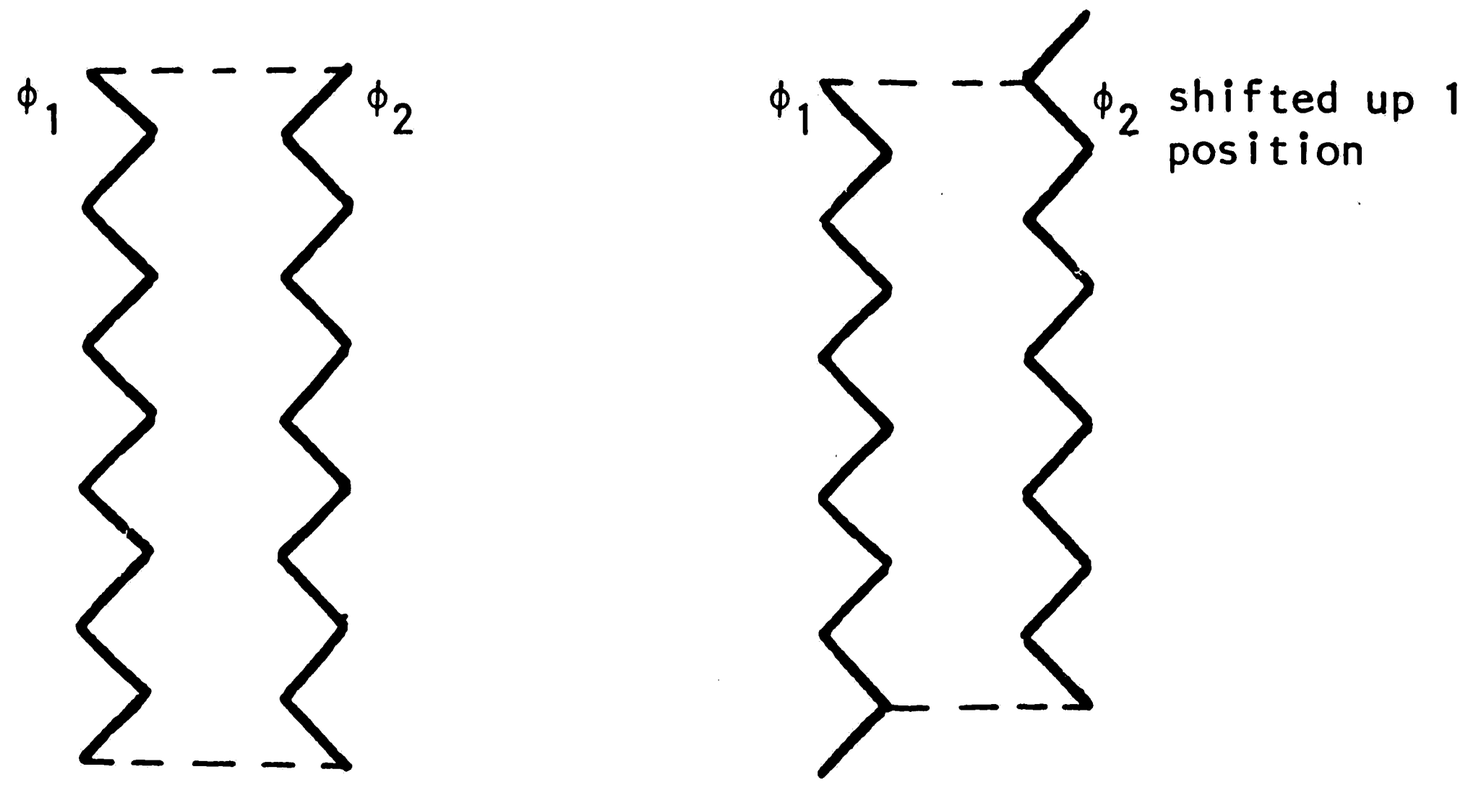


Figure 5.25 Two jagged vertical curves.

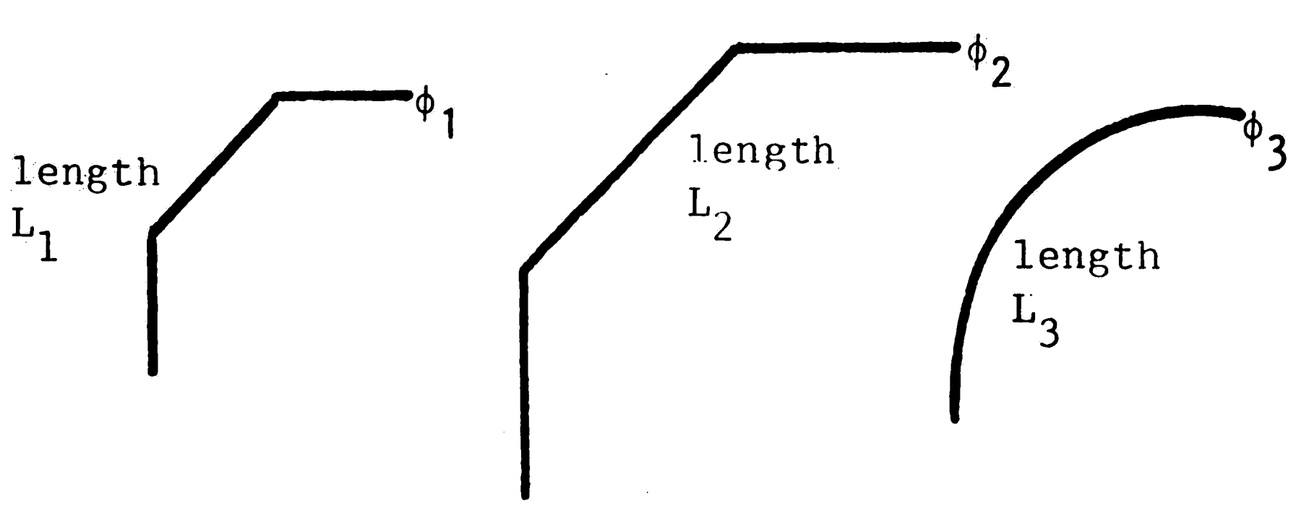


Figure 5.26 Three curves with different lengths.

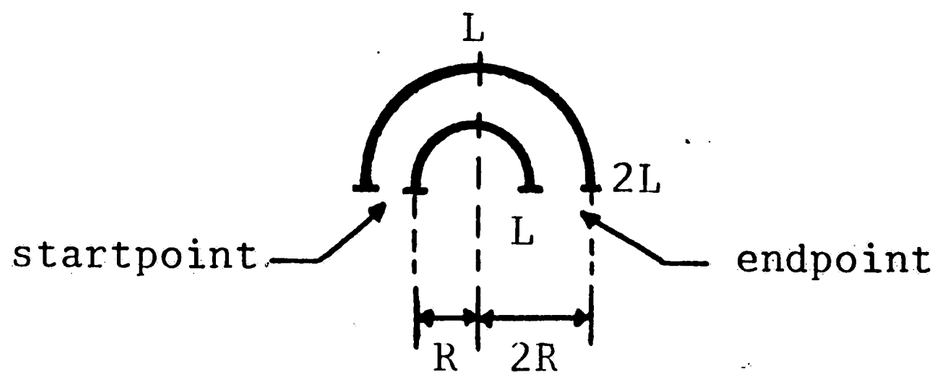


Figure 5.27 Two curves in the form of a semicircle.

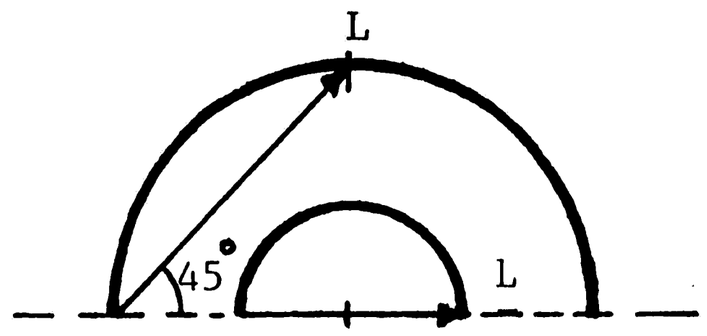


Figure 5.28 One element approximation of first part with length L of curves.

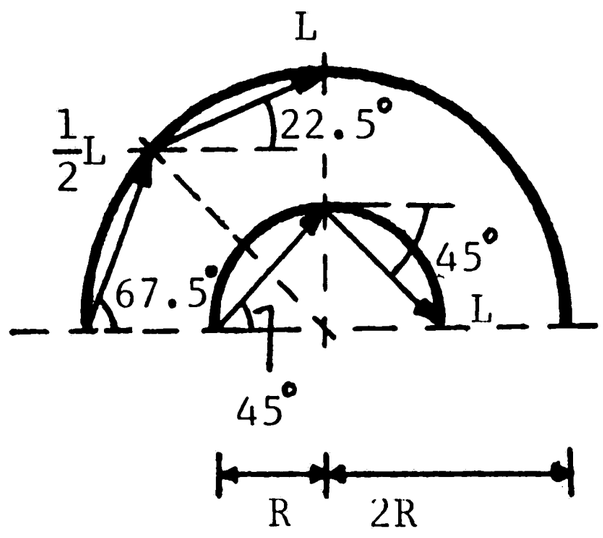
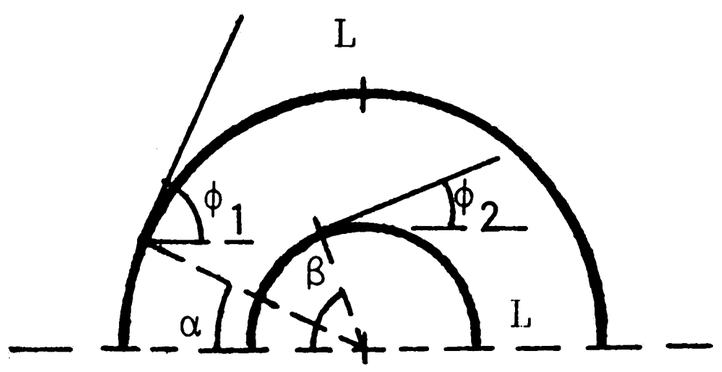


Figure 5.29 Two element approximation of first part with length L of curves.



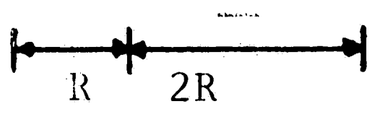


Figure 5.30 Infinitesimal approximation of first part with length L of both curves.

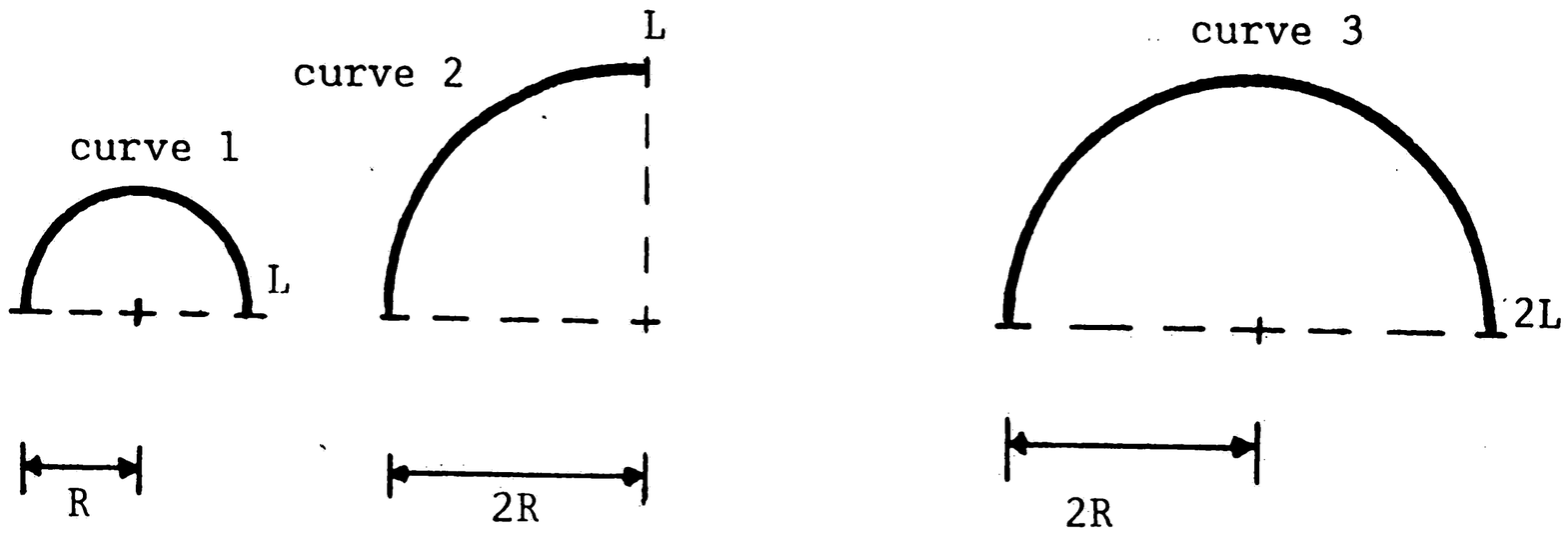


Figure 5.31 Three different curves, all part of a circle.

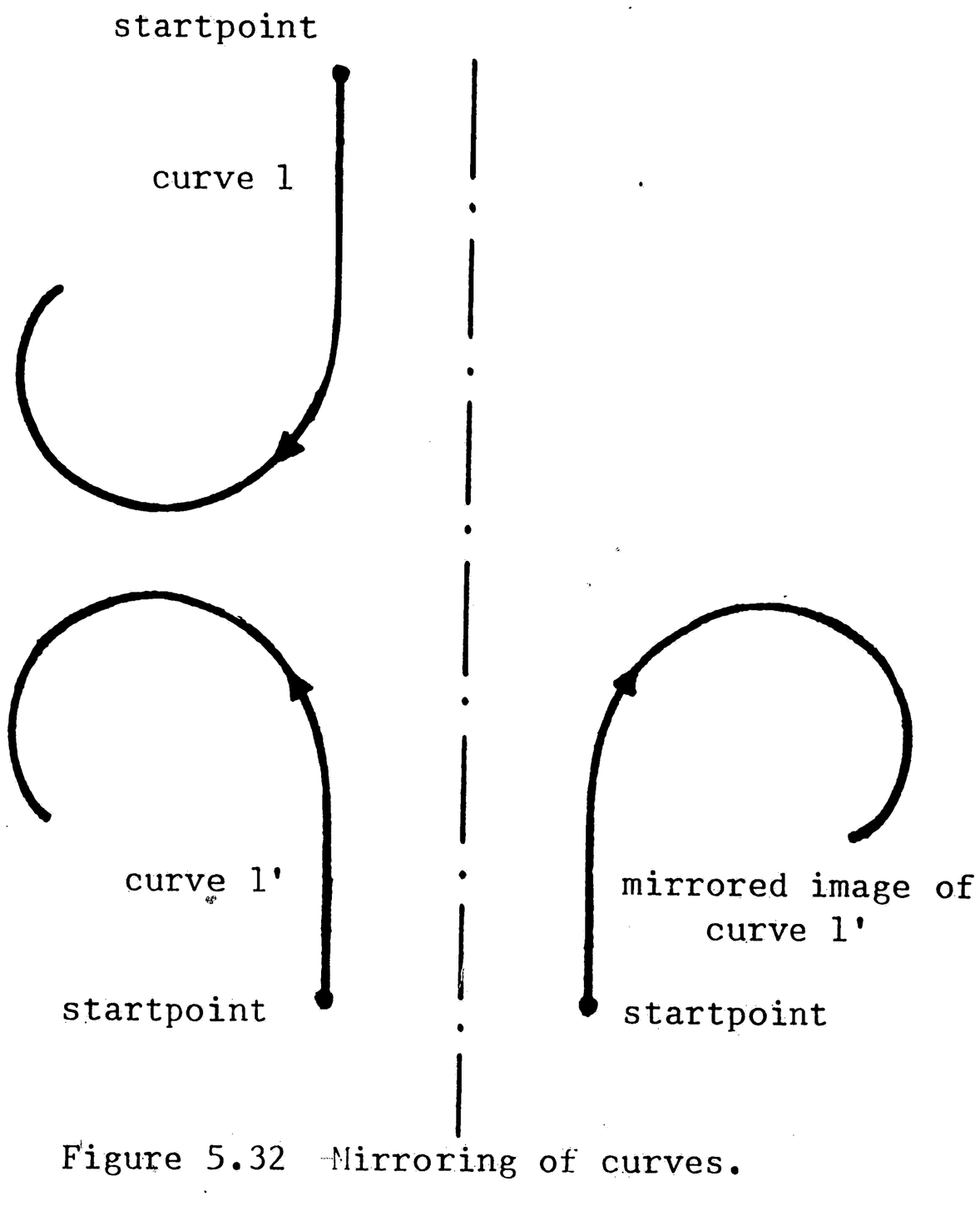
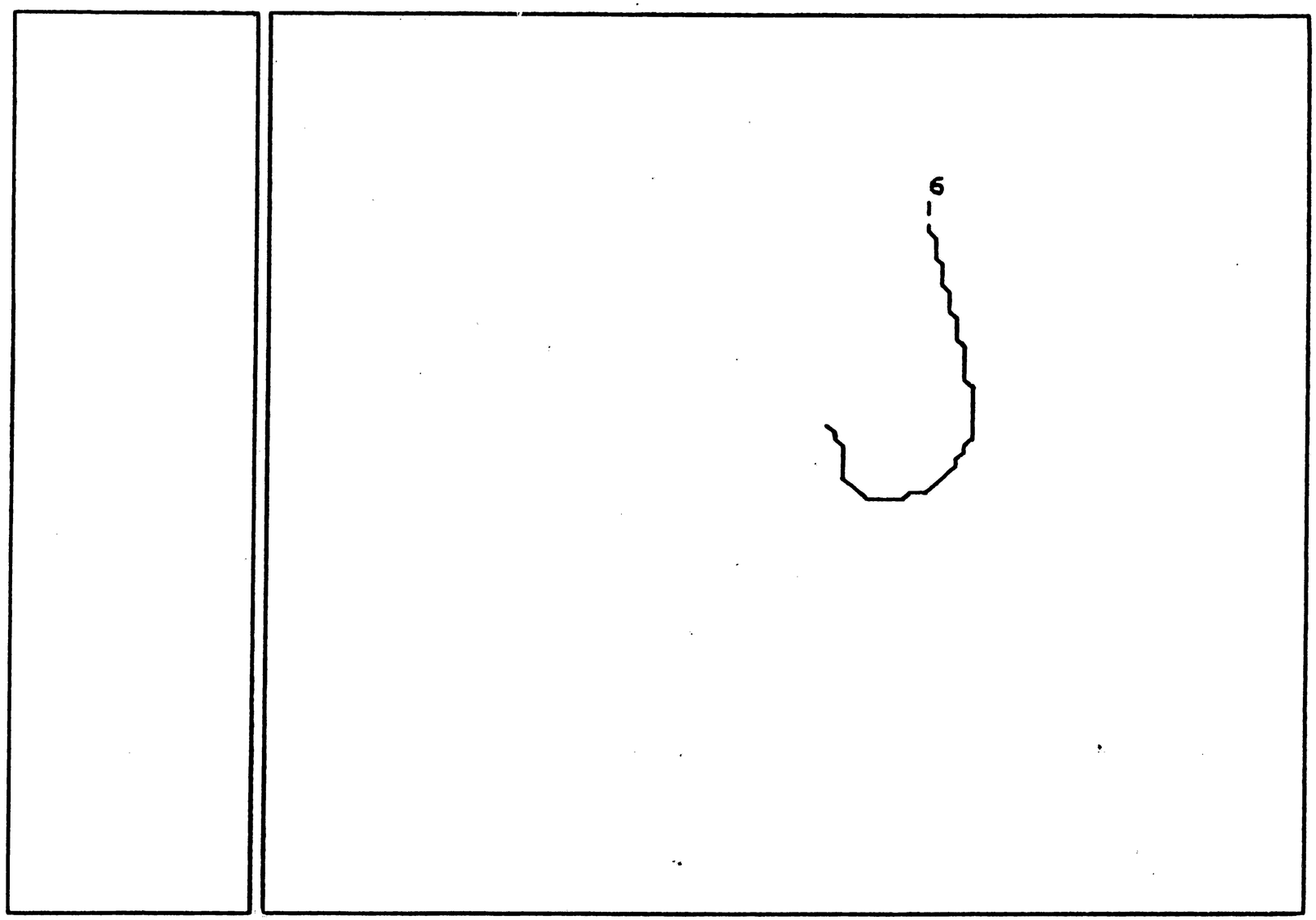


Figure 5.32 Mirroring of curves.



THE SIMILARITY VALUE IS: 1.0000
(HIT ANY KEY TO CONTINUE)

Figure 5.33 Similarity between a curve and itself.

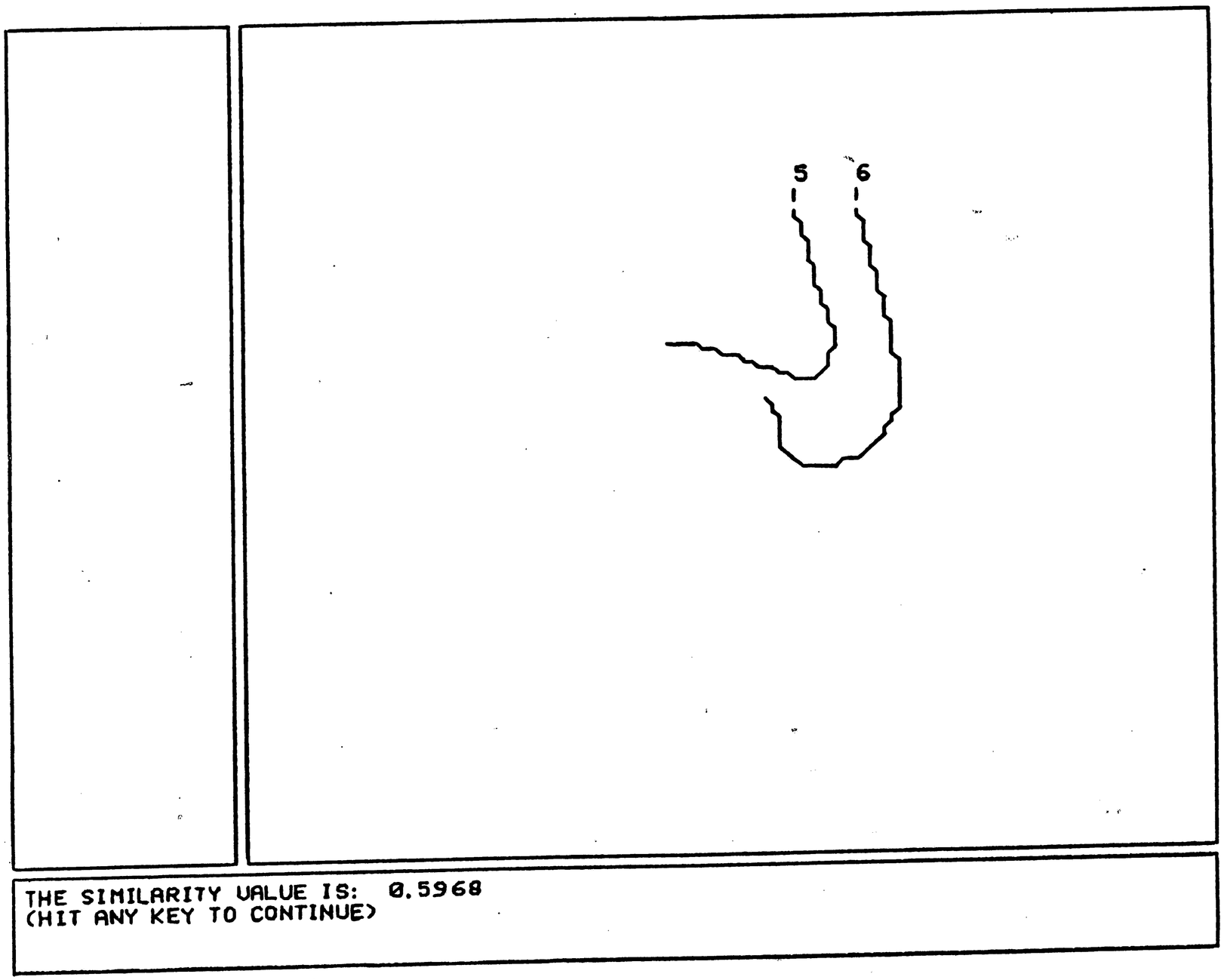
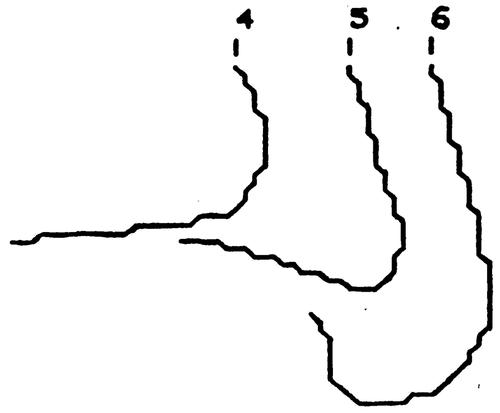
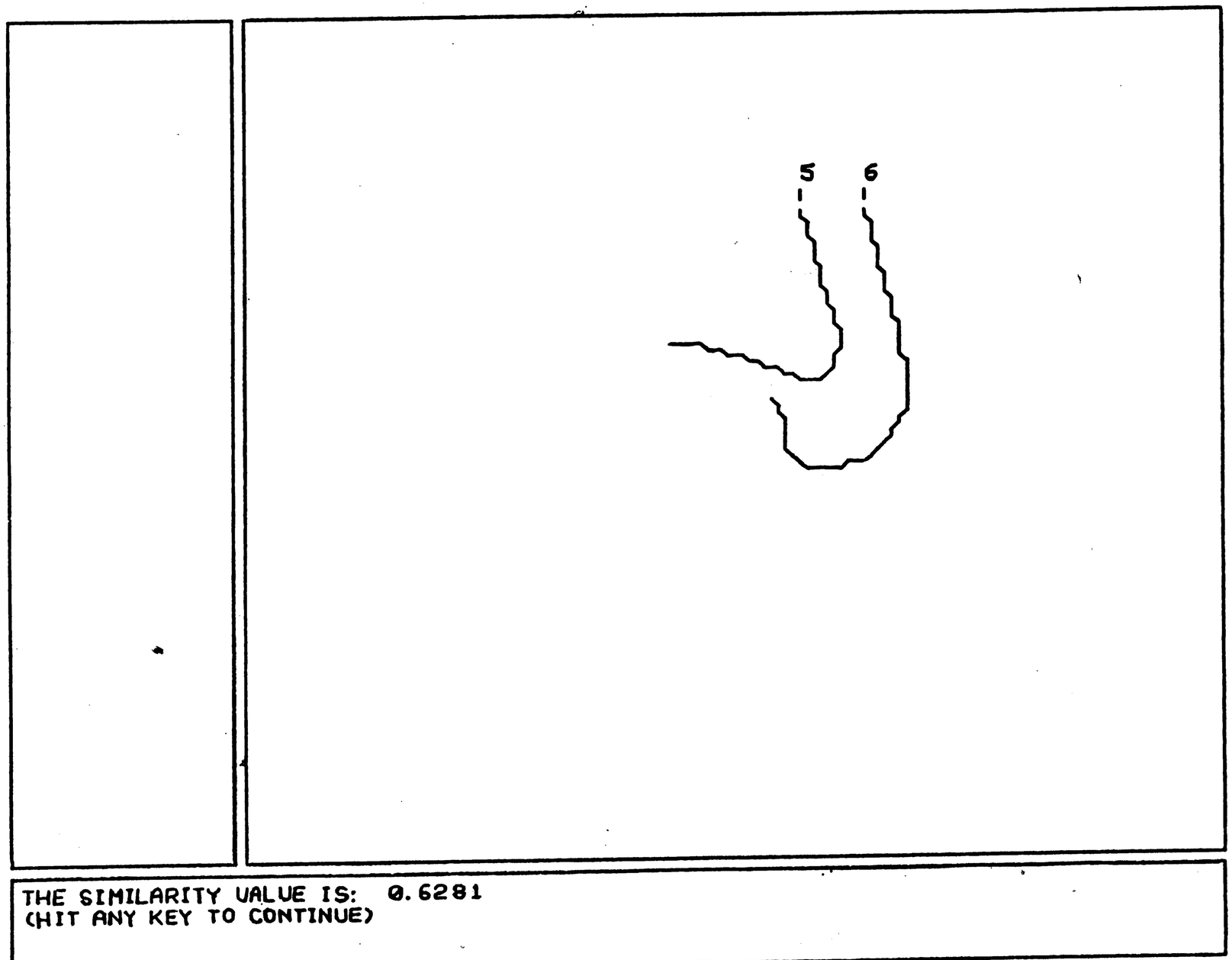


Figure 5.34 Similarity between two curves.



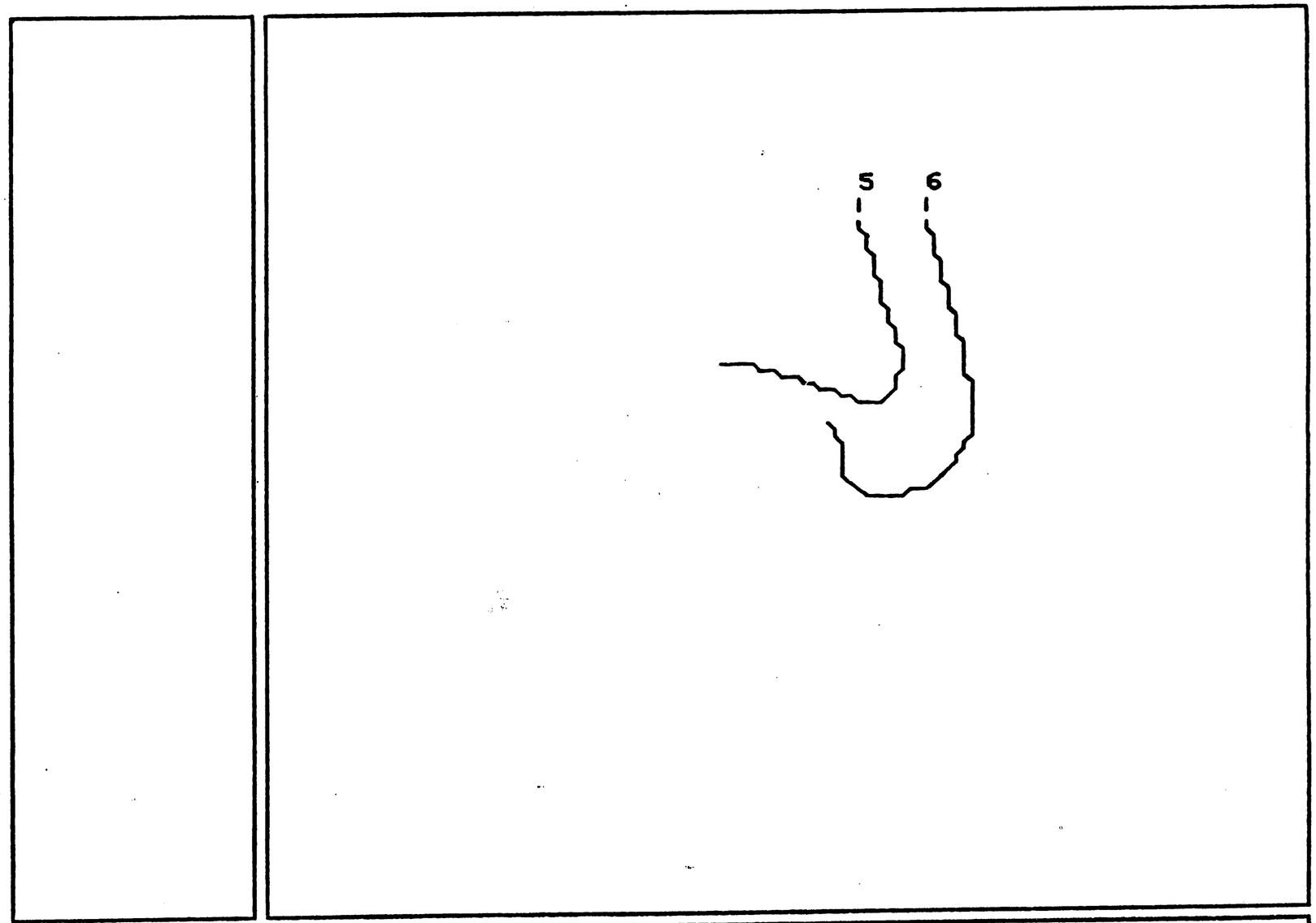
THE MEAN SIMILARITY VALUE FOR THIS CURVE IS: 0.5278
WITH A STANDARD DEVIATION OF: 0.0975
(HIT ANY KEY TO CONTINUE)

Figure 5.35 Mean similarity value (and standard deviation) for a set of 3 curves.



THE SIMILARITY VALUE IS: 0.6281
(HIT ANY KEY TO CONTINUE)

Figure 5.36 Similarity between two curves (using a shift range of 20%) applied to the curves of Figure 5.34.



THE SIMILARITY VALUE IS: 0.5820
(HIT ANY KEY TO CONTINUE)

Figure 5.37 Similarity between two curves (using the shortest curve length) applied to the curves of Figure 5.34.

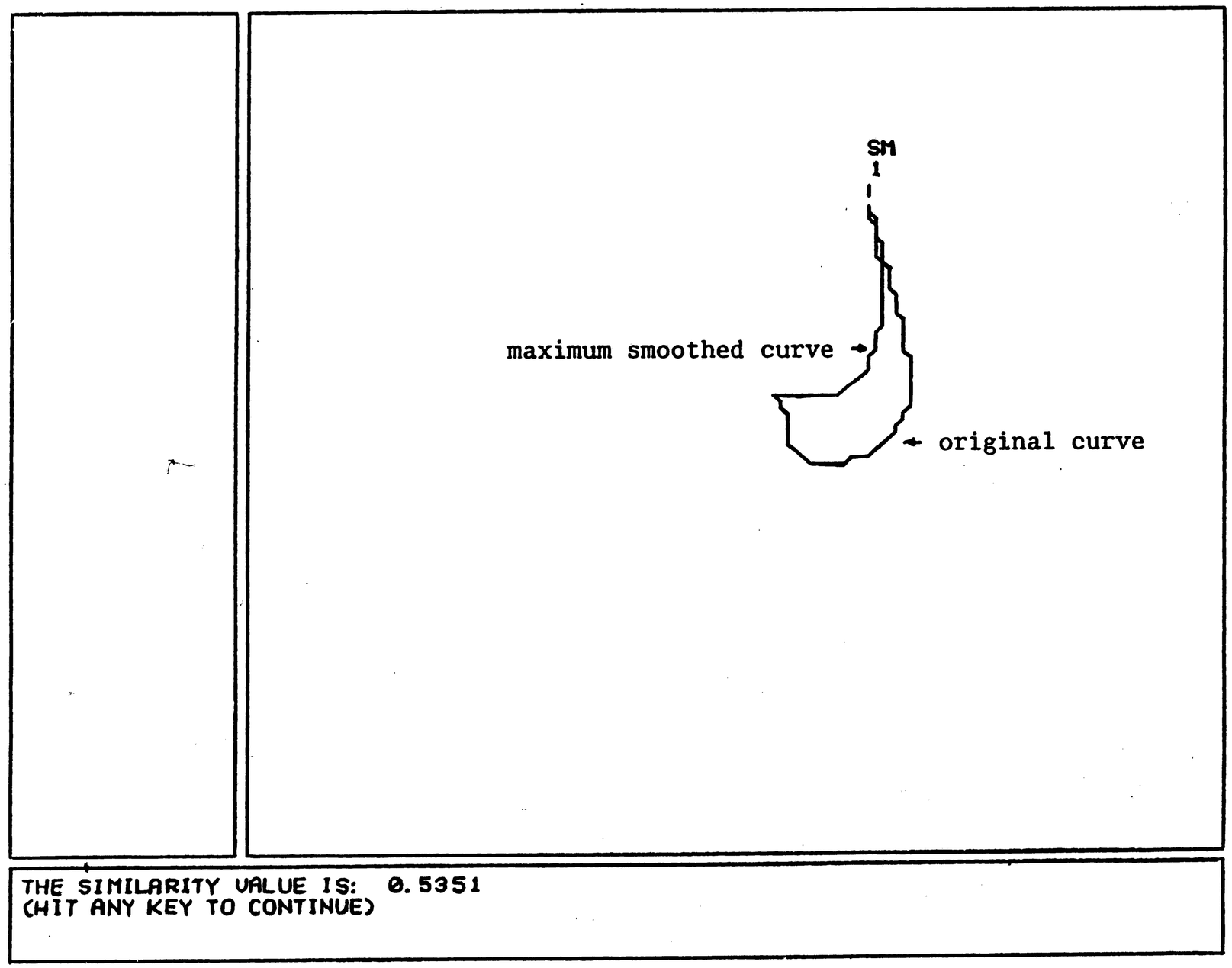


Figure 5.38 Similarity between original and smoothed curve.

REFERENCES

1. Carlson, A. B., Communication Systems, 2nd ed., McGraw-Hill, 1975.

2. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations", IRE Trans. on Electronic Computers, Vol. EC-10, June 1961, pp. 260-268.

3. Granlund, G. H., "Fourier Preprocessing for Hand Print Character Recognition", IEEE Trans. on Computers, February 1972, pp. 195-201.

4. Gumas, C. C., A General Pattern Recognition Technique for Open Curves, Master of Science Thesis, Department of Computer Science and Electrical Engineering, Lehigh University, 1985.

5. Kerstens, Pieter J. M., Manual for Spatave: An Image Processing and Pattern Recognition Program for Fluid Flow Analysis, Department of Mechanical Engineering, Lehigh University, September, 1985.

6. Lerner, E. J., "Sleuthing by Computer", IEEE Spectrum, July 1983, pp. 44-49.

7. Ozsoy, T. M., Bhalla, S., Summer, R., VS11 Graphics Package Reference and Example Manuals, CAD Laboratory, Lehigh University, 1983.

8. Pavlidis, T., Algorithms for Graphics and Image Processing, Computer Science Press Inc., 1982.

9. Rao, K., Balck, K., "Type Classification of Fingerprints: A Syntactic Approach", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 3, May 1980, pp. 223-231.

10. Zahn, C. T., Roskies, R. Z., "Fourier Descriptors for Plane Closed Curves", IEEE Trans. on Computers, Vol. C-21, No. 3, March 1972, pp. 269-281.

APPENDIX A: SPATAVE, AN INTERACTIVE IMAGE PROCESSING AND PATTERN RECOGNITION PROGRAM FOR THE ANALYSIS OF FLUID FLOW PATTERNS

In this appendix, the main structure of the program "SPATAVE" (for SPATial AVEraging) is presented. For a more detailed description, see Kerstens (1985). The program is written in FORTRAN 77 and runs on a VAX 11/780 computer equipped with VS11 color monitors. For this program the display is divided into three parts (so-called windows). The first window has a yellow border and is displayed in the upper right corner. This window will display all the curves (original, averaged, and smoothed curves). The second window is positioned at the bottom of the display and has a blue border. In this window all program messages will be displayed. The third window also has a blue border and is positioned at the left of the first window. This window is for future use (display of additional data or menus). There is also an invisible fourth window (with no border). This is used for full screen displays (in which case the curves are shown on the whole screen and the other windows are removed from the screen). The windows are shown in Figure A1.

The set-up of the program is best understood by looking at Figures A2, A3, and A4. Although these flow charts are not exhaustive, they do contain all the necessary information. The flow chart of the main routine is displayed in Figure A2. The

initialization block sets a number of variables to their initial values and calculates some basic parameters for each curve that is an input. After some scaling (the user can pick the scaling factor and the center of display) the program will ask the user if he wants to display complete frames (a frame is a single image containing several curves and is obtained from one video image taken at a particular point in time). If the user decides to use this option, the program will execute routine A. The program will now ask the user if he wants to display just a single curve. Again the user can decide to do so in which case the program will run routine A'. The next option that is offered is averaging of complete frames. The following option is averaging of a single curve on several different frames. The final option makes it possible for the user to average an arbitrary set of curves that can, but do not have to be, on a single frame. Finally, the user can jump back to the beginning of the program and do some further processing.

The program is set up to give the user maximum flexibility while still making it possible to get some quick results. In fact, speed of operation decreases if one goes down in the flow chart while flexibility increases. So if one wants to average complete frames, the results can be obtained quickly. However, if one wants to pick each of the curves that should be averaged (increased flexibility), one has to do so by selecting the curves one by one (decreased speed).

Figure A3 shows how the frames and curves are displayed (routines A and A'). Among other things one can clear the window, change the color of the next curve/frame (in fact change the shade of green), and do a similarity test. This can be repeated as many times as desired.

Figure A4 shows how the frames and curves are averaged. Among other things one can again clear the window, average the curves/frames, smooth the curves/frames, and calculate similarity values by picking some of the displayed curves. Again, this can be repeated as many times as desired.

The reader should note that any of the following can be done with the program SPATAVE in its current state:

- display any frame(s) (with or without clearing the screen)
- display any curve(s) (with or without clearing the screen)
- smooth any curve(s) (original or averaged)
- average any of the original curves (on the same or on different frames)
- calculate the correlation/similarity value between any of the displayed curves/frames (original, smoothed, or averaged)

With these capabilities, the program SPATAVE should be a useful tool in the analysis of fluid flows.

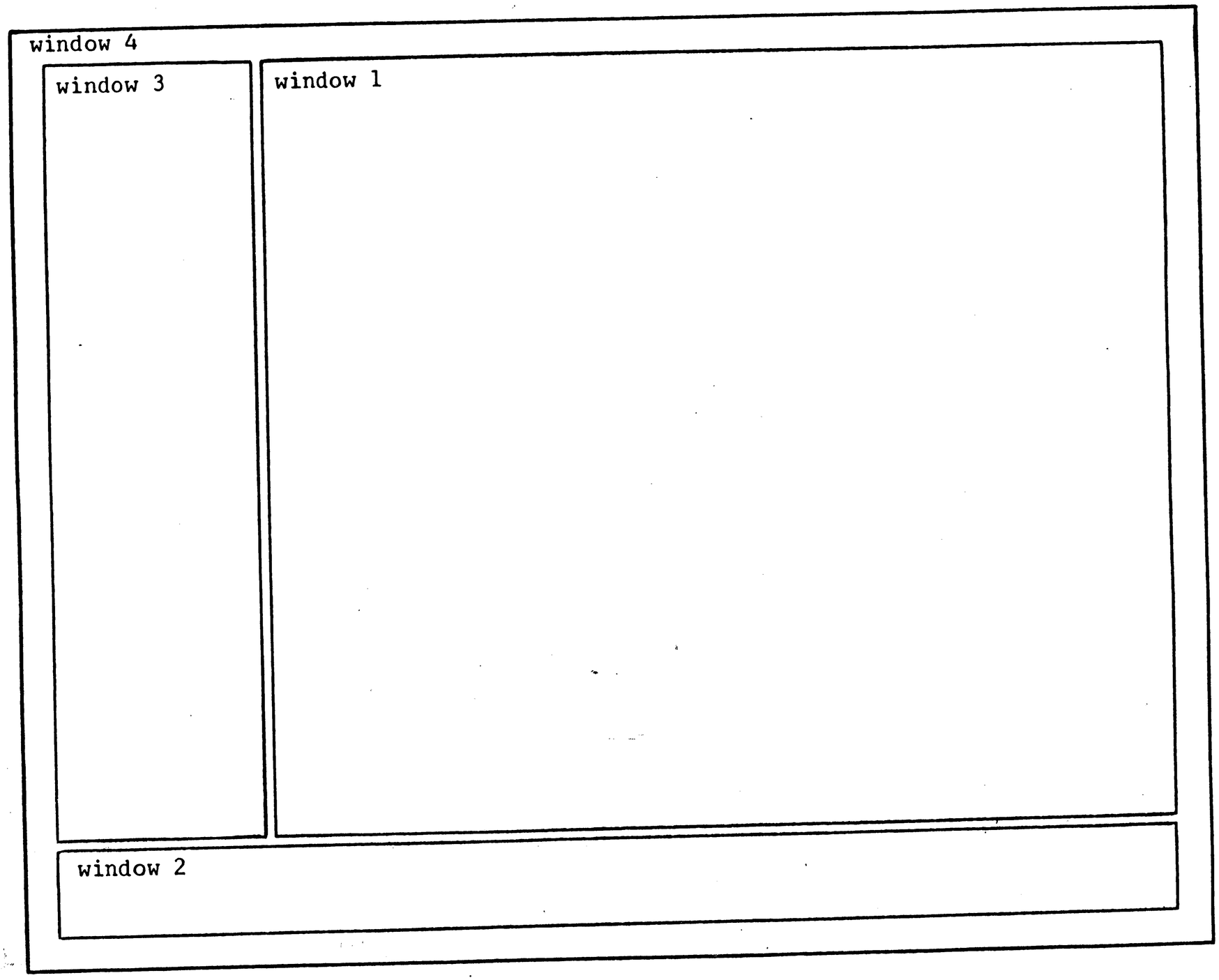


Figure A1 Display setup.

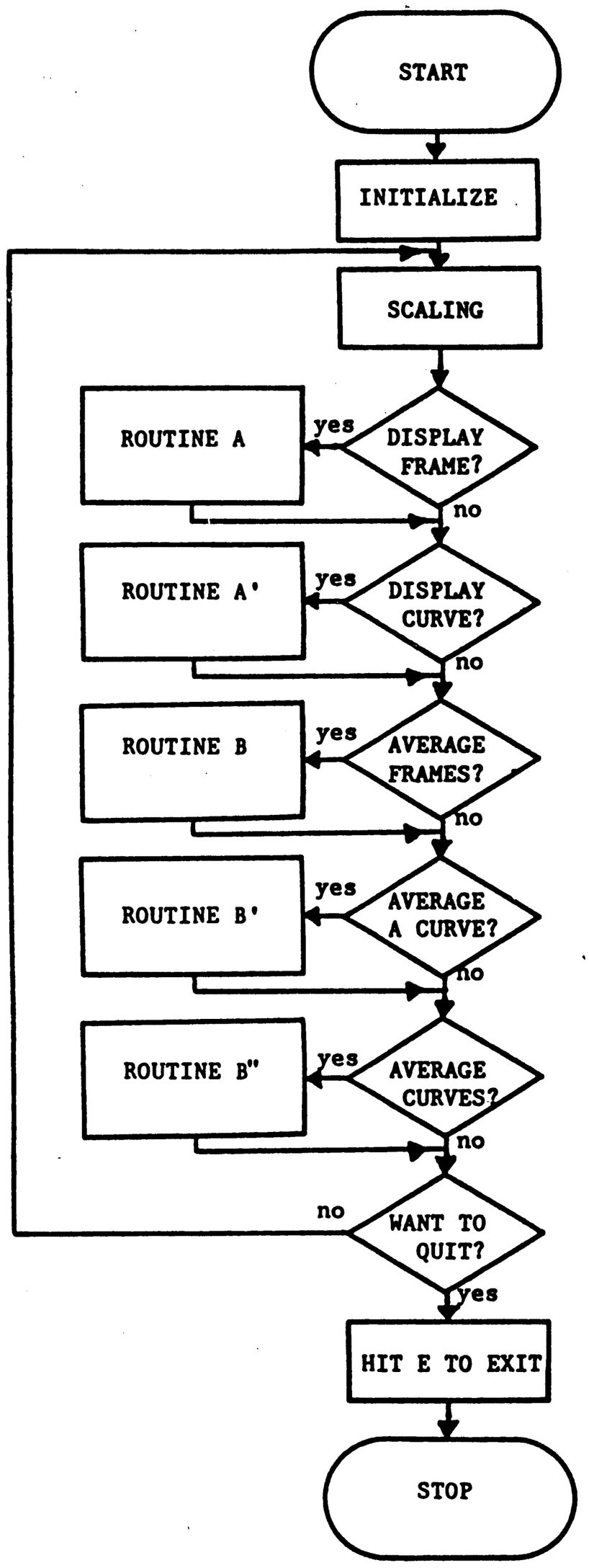


Figure A2 Flow chart of the main routine.

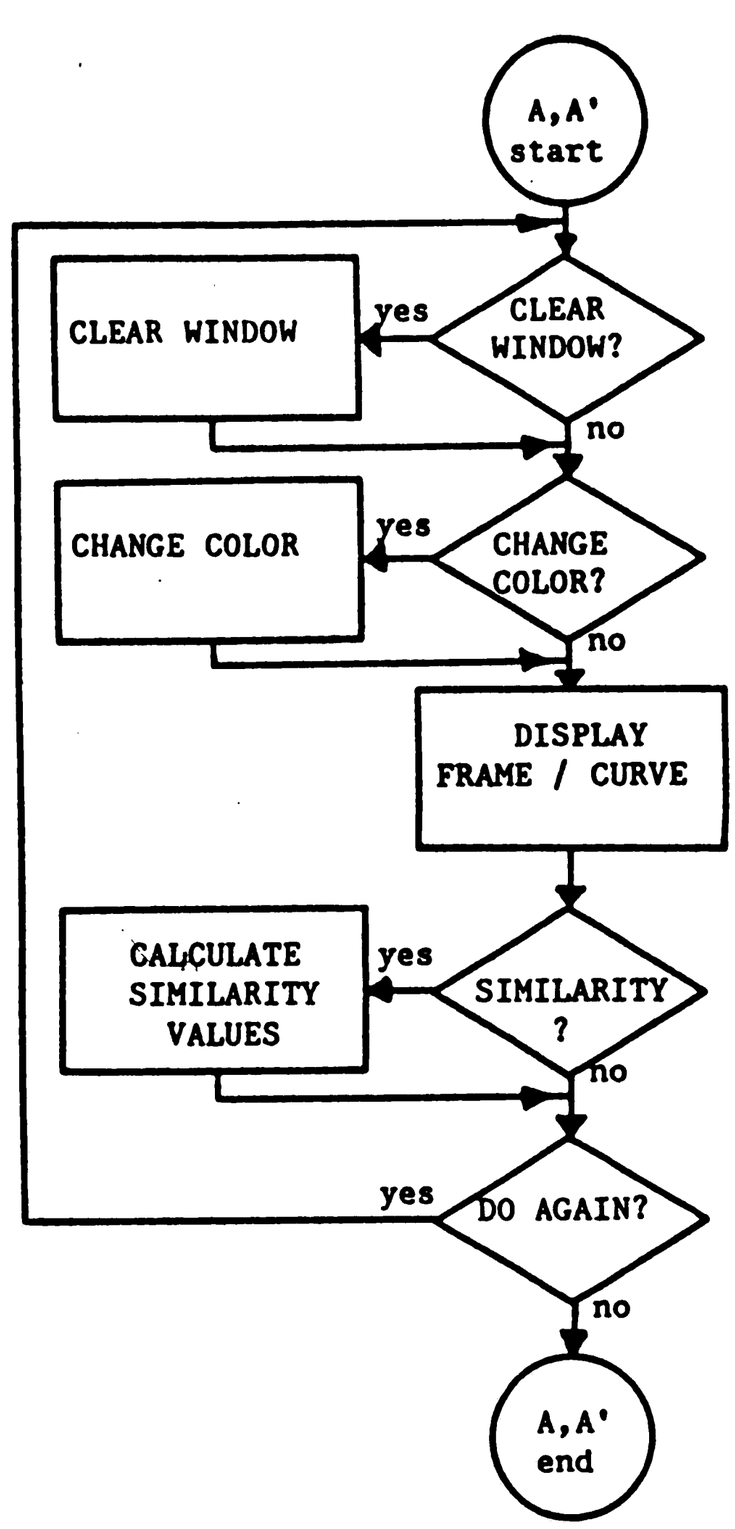


Figure A3 Flow chart of routines A and A'.

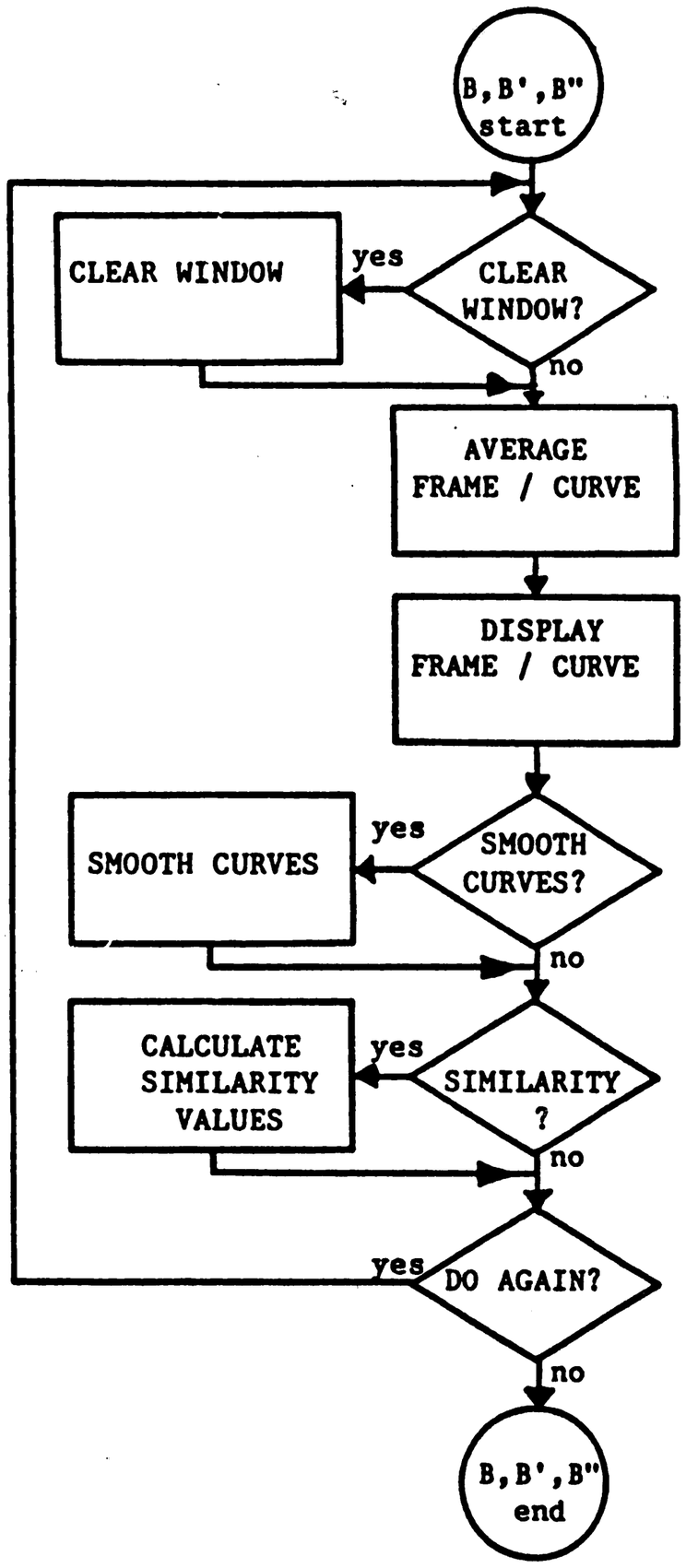


Figure A4 Flow chart of routines B, B', and B".

APPENDIX B: Author's Biography

Pieter J. M. Kerstens received his Kandidaats degree and his Ir. degree in Electrical Engineering from Eindhoven University of Technology, The Netherlands, in 1981 and 1983 respectively. His thesis work included digital satellite transmissions in the 11 and 14 GHz bands, with the Orbital Test Satellite. These transmissions were the first digital satellite transmissions in The Netherlands.

During his graduate studies, the author also worked on fiber optic transmission and a three month project at the Israel Institute of Technology, Haifa, Israel. In 1983 he joined Philips Laboratories in Briarcliff Manor, New York, where he was engaged in repeaterless long wavelength and single mode fiber optic transmissions of FM video signals. This work resulted in a paper that he presented at the NCTA Conference, Las Vegas, Nevada, in June, 1985. He is also author or co-author of several technical reports. The author's current interests are in robotics, flexible automation, communication, optimal control, and image processing.