Feature Point Detection and Curve Approximation for Early Processing of Free-Hand Sketches

by

Tevfik Metin Sezgin

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by ....
Randall Davis
Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Feature Point Detection and Curve Approximation for Early Processing of Free-Hand Sketches

by

Tevfik Metin Sezgin

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Freehand sketching is both a natural and crucial part of design, yet is unsupported by current design automation software. We are working to combine the flexibility and ease of use of paper and pencil with the processing power of a computer to produce a design environment that feels as natural as paper, yet is considerably smarter. One of the most basic steps in accomplishing this is converting the original digitized pen strokes in the sketch into the intended geometric objects using feature point detection and approximation. We demonstrate how multiple sources of information can be combined for feature detection in strokes and apply this technique using two approaches to signal processing, one using simple average based thresholding and a second using scale space.

Thesis Supervisor: Randall Davis
Title: Department of Electrical Engineering and Computer Science

Acknowledgments

- I would like to thank my thesis advisor Prof. Randall Davis for his support and guidance during my time at the AI LAB. I have particularly appreciated his availability, patience and feedback.
- I would like to thank my parents Fatin Sezgin, Hatice Sezgin, and my sister Zerrin Sezgin for their emotional support and encouragement.
- I would like to thank Ford Motor Company and sponsors of Project Oxygen for the financial support.
- Finally I would like to thank my friends in Boston and Syracuse, my officemates Mike Oltmans, Mark Foltz, and wonderful people in our research group for their encouragement and advice on everything from housing to random bits of wisdom on LaTeXformatting.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Making sense of sketches depends crucially on the ability to capture and interpret what hand-drawn strokes are meant to represent. We demonstrate how multiple sources of information can be combined for feature detection in strokes and apply this technique using two approaches to signal processing, one using simple average based thresholding and a second using scale space.

This chapter introduces the problem of sketch understanding and explains why stroke approximation (i.e., approximation of free-hand drawings with low level geometric objects) is needed. Chapter two states some of the problems encountered while dealing with hand drawn strokes. Chapter three illustrates how speed and curvature data can be used for feature detection. We describe two methods for finding the minima of speed and maxima of curvature in the presence of noise. We also compare these two methods, focusing on tradeoffs in feature point detection quality, algorithmic complexity and speed.

In chapters four and five we introduce a method for combining fits generated by multiple sources of information, each using different methods for feature point detection, and describe a framework for dealing with strokes containing curves as well as straight lines.

Next we describe how free-hand strokes can be classified as primitive geometric objects via template matching, and demonstrate how a higher level recognition system can be built using our system. We describe how a sketch understanding system for early design stages of mechanical systems can be built on top of the system described here. Finally we review related work and future directions that can be taken from this work.

1.1 Sketching in the Early Design Phases

Computers are used extensively in current design practice. Design automation tools are used by engineers from almost all disciplines. These tools are especially well suited for the later phases of the design, where the focus is more on implementation details rather than the conceptual aspects of the design. For example, in the domain of software engineering, conceptual design and brainstorming is usually done on paper, or on a white-board, while implementation is done using a software development environment for a particular programming language and platform. In the case of mechanical engineering, the conceptual structure of the mechanical devices to be built are sketched on paper, and later in the implementation phase, CAD programs are utilized for detailed design.

These examples illustrate possible uses of sketching in the design process, but unfortunately there is very little support for sketching, despite its role in conveying ideas, guiding the thought process, and serving as documentation[20].

Even crude sketches say a lot about the conceptual structure, organization, or physical description of the system being designed right from the beginning of the design process. For example, the sketch in Fig 1-1 (part of the direction reversing mechanism in a walkman) contains information about the rough geometry of the mechanism. It shows what components there are, and how they are connected.

Although sketches are an integral part of early design, when the design process moves into digital media, sketches and diagrams are usually left behind. The information about the rough geometry, components and information about how components are connected are left behind with the sketch. The designer opens a blank CAD screen and starts building the device being designed piece by piece. We believe some of the time and effort spent during the transition from early design/brainstorming

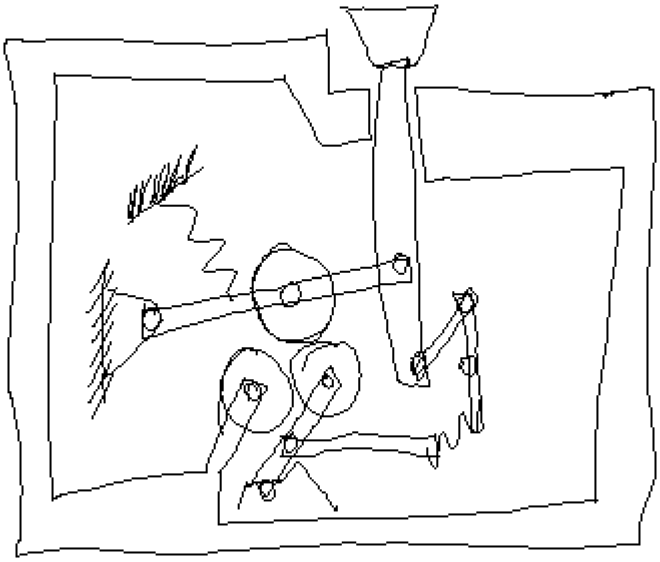


Figure 1-1: The rough sketch of the direction reversal mechanism of a walkman illustrating our notion of an informal sketch.

phase can be reduced by making it possible to capture the sketches and diagrams in digital environment.

By capturing a sketch we mean understanding the geometry of the components present in the sketch. For example, in Fig. 1-1 we can identify components such as rods (represented by rectangular shapes), pin joints and gears (represented by circles), two connections to ground (represented by a number of parallel lines), springs, and the casing of the mechanism. A description in terms of the geometry and position of the components present in a sketch is far more powerful and meaningful than a simple bitmap image.

Sketching as a means of creating designs removes heavy dependence on menu or toolbar based interfaces that require the user to work with a fixed vocabulary of shapes and build more complex geometries by combining simpler ones. For example, if the user wants to create a complex shape as in Fig. 1-2 using a traditional CAD tool, he or she has to go through menus, pick the right tool for drawing lines, draw the linear parts of the shape, select the tool for specifying curves and draw the curved regions by specifying control points. Then the positions of the control points specifying the curve should be adjusted to achieve the desired shape. On the other hand, the system we describe in this thesis makes this task as simple as taking a pen and drawing the desired shape, requiring no menu selections.

Some researchers suggested Graffiti and gestures to remedy some of the problems associated with menu based interfaces. Unlike Graffiti Our system allows users to draw in an unrestricted fashion. For example, it is possible to draw a rectangle clockwise or counterclockwise, or with multiple strokes. Even more generally, the system, like people, responds to how an object looks (e.g., like a rectangle), not how it was drawn. This is unlike Graffiti and other gesture-based systems such as [10], and [17] where constrained pen motions like an L-shaped stroke, or a rectangular stroke drawn in a particular fashion is used to indicate a rectangle. This, we believe, produces a sketching interface that feels much more natural.

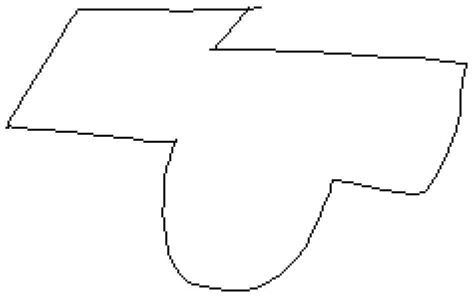


Figure 1-2: A complex shape.

1.2 The sketch understanding task

The sketch understanding task itself is a very hard problem. We believe that an intelligent online sketch understanding system should have a combination of the following features:

- Stroke approximation and recognition: This refers to a low level system that takes input strokes, classifies them as members of low level geometric primitives (such as lines, ovals, polylines, curves, as well as their combinations) and recognizes strokes that belong to special subclasses (such as rectangles, triangles, circles).

- Ambiguity resolution: Sketches are inherently ambiguous. For example, depending on the context, a circle may be interpreted as a pin joint or a circular body. The ambiguity resolution module takes the output of the stroke approximation and recognition layer as input and resolves the ambiguities making use of domain specific knowledge. Work in [1] describes such a system.

- Speech input: It is impossible to resolve all the ambiguities in a sketch without any user input. Even humans can't be expected to resolve all ambiguities without any further explanations Of course we want to do this in a minimally intrusive manner for the users. Speech input is a natural choice for this task. The user can verbally give more information about the sketch, or in the case of mechanical engineering design, the user can help the system resolve ambiguities by describing the structure or behavior of the device in question. [13].

Above, we have itemized some of the key features that sketch understanding systems should have. This thesis describes a system for the stroke approximation and recognition tasks mentioned above.

1.3 Input

The input to our system is an array of time-stamped pixel positions digitized by a tablet. Having the timing data allows us to derive useful properties of strokes such as pen's speed and acceleration. By definition a stroke is an array of (x, y, t) values that describe the path that the pen traces between mouse down and mouse up events.

1.4 Output

The output of the system is a geometric primitive that approximates the input stroke. The geometric primitives we support are lines, ovals, polylines, curves, and complex shapes consisting of curves and polylines. These primitives cover all possible shapes. Supporting a broad range of shapes proves to be useful in complex domains such as mechanical engineering sketches. Fig. 1-3 illustrates the output we produce for the sketch in Fig. 1-1. In this figure, the input strokes are approximated by geometric primitives and some domain specific objects (e.g., gears, springs, connections to ground) are recognized by the higher level recognizer we built.

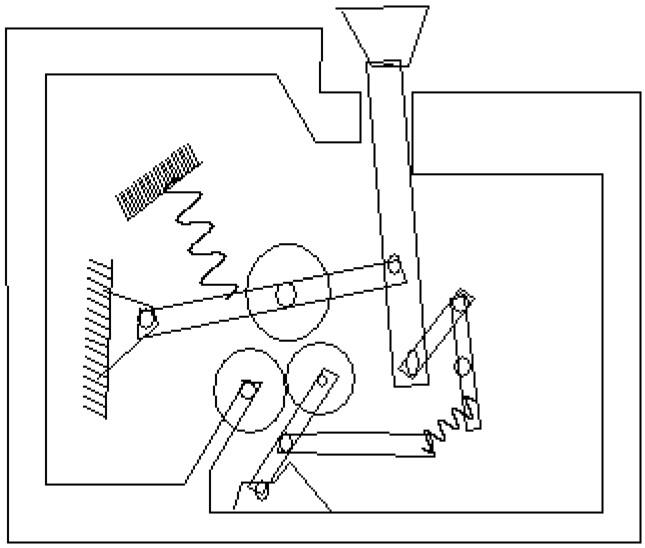


Figure 1-3: The output produced by our algorithm for the sketch in Fig. 1-1.

Chapter 2

Primary Issues

There are several problems that complicate stroke approximation and sketch recognition process. To set the context for the discussion, we illustrate with an example the distinction between a sketch and a diagram. By a sketch we mean a crudely drawn, messy looking, freehand diagram (as in Fig 2-1), while by diagrams we mean clean figures, carefully drawn (Fig 2-2). The difference in character between these two figures produces a number of difficulties when working with sketches.

2.1 Imprecision due to freehand sketching

Geometric primitives such as circles, lines or rectangles, are typically characterized by properties such as radii, center, height, width or vertex positions. This is a highly idealized abstract view of these objects, and sketching a circle of fixed radius or a line with a constant slope is in practice beyond most people's capability. This means we can't depend on uniformity in strokes for feature detection.

2.2 Digitization noise

The input to our system is a stream of points digitized by an LCD tablet. As usual, digitization results in loss of information. Digitization-related problems show up in two forms in the context of stroke approximation:

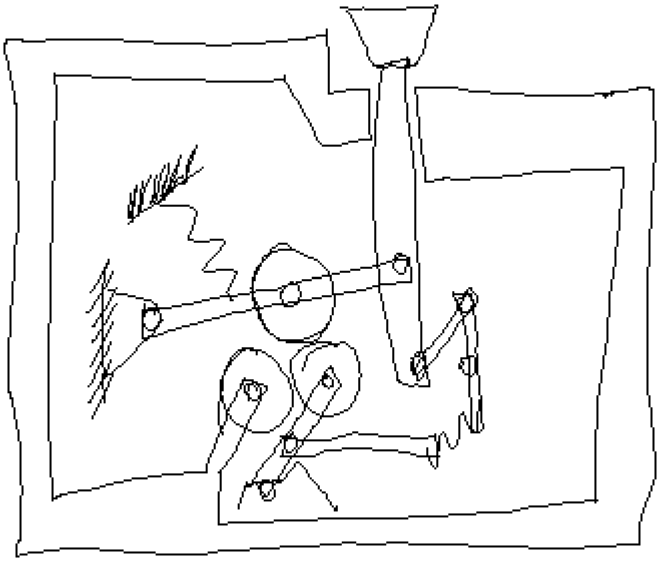


Figure 2-1: The rough sketch of the direction reversal mechanism of a walkman illustrating our notion of an informal sketch.

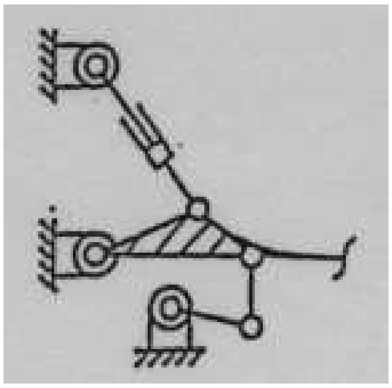


Figure 2-2: A carefully drawn diagram. Compare to Fig. 2-1.

- Pen's physical position on the tablet, a continuous function in time, is mapped to discrete (x, y) points in screen coordinates, thus there is an information loss due to digitization even if we assume ideal conditions where the exact physical position of the pen is assumed to be known at every moment.

- Noise due to imprecision in localizing pen position is yet another problem. In the particular type of digitizing tablet that we were using, straight lines could not be drawn even with a ruler, because behavior of the capture device is unspecified when the pen position is in between two pixels. Fortunately the

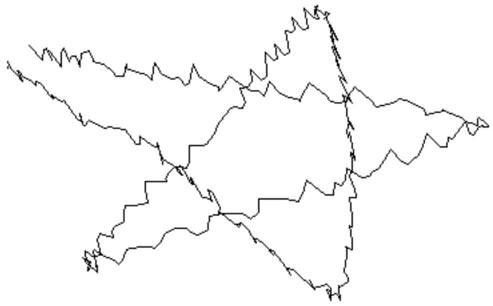


Figure 2-3: A very noisy stroke.

noise introduced in this fashion is only a few pixels wide for freehand sketching for the tablet we were using. Other tablets, however, introduce noise that is larger in magnitude. For example, one of the tablets we tried out for evaluation picked up noise from the computer monitor. Fig. 2-3 shows a star shaped stroke captured with this tablet when it was very close to the monitor.

2.3 Extracting direction information

Before discussing how we derive the direction[1] information, we point out some of the issues in capturing freehand strokes via digitizing tablets.

An unwanted side effect of digitizing pen position comes up when we try to extract the direction along the curve. There are several ways of computing the tangent to the curve. Depending on which method we use, the direction data we compute changes. The simplest way of estimating the tangent at a point is to look at the relative position changes Δx and Δy between consecutive data points, and approximate the slope of the tangent between those points by $\Delta y/\Delta x$. The direction is then obtained by $\theta = atan(\Delta y, \Delta x)$ [2]. Although this sounds like a natural way of defining the tangent and computing direction, it fails if data points are very close to one another. For example, as illustrated in Fig. 2-4 if we use the relative position changes between

[1]In the rest of this document, by direction we refer to the angle between the tangent to the curve at a point and the x axis rather than the slope of the tangent line because of the singularities in deriving slopes (for vertical lines).

[2]$atan(y, x) : \Re x \Re \rightarrow (-\pi, \pi]$ is a variant of arctangent function that takes the signs of each of its arguments into account.

consecutive data points to compute the direction for two points 3 pixels away from one another, we will be able to compute only 16 distinct angles in the $(-\pi, \pi]$ range. The problem gets worse as the separation between the points becomes smaller. The Phoenix system [18] tries to deal with this problem by preprocessing the input so that consecutive points are at least δ pixels apart from one another for some sufficiently large δ, so that the interval $(-\pi, \pi]$ is covered by more angles. This means some points are discarded, resulting in loss of information.

Another problem in dealing with digitized freehand strokes is that of noise (this problem is also present for scanned documents). The naive approach of deriving the direction data using slopes of lines connecting consecutive points ends up being too noisy to be informative. One approach frequently used in dealing with this noise is smoothing the direction data, perhaps by convolving with a Gaussian filter. Due to lack of sufficient data points and sparseness of the data points, this degrades the performance of corner detection by treating corners as noise to be smoothed.

Unlike scanned drawings, in our system data points may be some distance from one another. The stylus typically travels numerous pixels between samplings, because while digitizing tablets have sub-millimeter accuracy of pen placement, they are typically not sampled fast enough to provide a data point every time the pen moves from one pixel to the next in a freehand sketch. During freehand sketching the stylus may reach speeds of 12-15 inches per second. With typical tablet sampling rates of 50Hz, the number of points sampled per inch drops down to only 4-5 ppi and points are sampled sparsely in screen coordinates. For example, the LCD tablet we used had an active area of 10.7x8.0 inches and a resolution of 1024x768 pixels, so with a sampling rate of 4-5 ppi, the stylus would move 20-25 pixels between samples. This lack of resolution compared to scanned images (that may have thousands of points per inch) means we have to show extra care in deriving direction information.

We chose to solve the problems above without discarding points or using Gaussian smoothing. We compute the direction at a point by fitting an orthogonal distance regression (ODR) line to a small window of points centered at the point in question. Orthogonal distance regression finds a line that minimizes the sum of the orthogonal

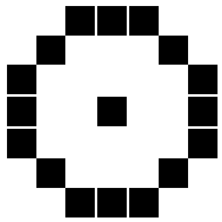


Figure 2-4: The outer circle is formed by all the points that are 3 units away from the point in the center. As seen here, if we use the relative position changes Δx and Δy between consecutive data points to compute the direction, the values we will be able to compute only 16 distinct angles in the $[-\pi, \pi]$ range.

distances from the points to the line (unlike linear regression, which minimizes only the y-distance). For computational efficiency we use a discrete approximation to the ODR that is good to 0.5 degree[3]. Our method requires choosing a neighborhood size $k = 2n + 1$ which covers the point in question, n preceding points and n following points. At the end points of the stroke where the neighborhood is not well defined for k, we use choose a smaller neighborhood to ensure that the window defining the neighborhood doesn't extend beyond the end points.

[3]Principal component analysis solves the same problem: the direction at a point is given by the eigenvector of the largest eigenvalue of the covariance matrix for the window of points surrounding the point in question. But this is computationally more expensive than our ODR approximation, which is more than accurate enough for our purposes. There are also gradient descent methods for ODR [7], but these don't provide any significant computational improvement.

Chapter 3

Feature Point Detection

This chapter illustrates how curvature and speed data can be used to detect feature points (i.e., corners) in a stroke. We begin by describing how we derive curvature and speed data for a stroke. Later we illustrate how we detect feature points using this data. Note that we are not simply trying to polygonalize the input stroke. We want to avoid representing curved portions of the input stroke via polygonal approximations because our curved region detection method depends on this.

3.1 Deriving stroke properties

As noted, stroke is a sequence of points along with timing information indicating when each point was sampled. Below, we explain how we derive curvature and speed information.

3.1.1 Curvature

Given direction data d, curvature is defined as $\partial d/\partial s$ where s is the accumulated length of the curve from the beginning to the point of interest. Note that curvature between points far from one another is smaller compared to two points with the same difference in direction that are closer together. This property of curvature makes it a more suitable indicator of corners than simple the pointwise change in direction, as

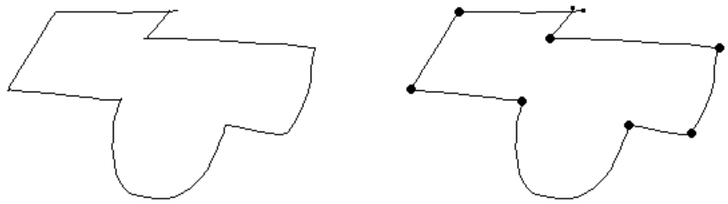


Figure 3-1: The stroke on the left contains both curves and straight line segments. The points we want to detect in the vertex detection phase are indicated with large dots in the figure on the right. The beginning and the end points are indicated with smaller dots.

data points are not spread uniformly (e.g., they are typically closer to one another around the corners).

3.1.2 Speed

We derive the instantaneous speed of the pen between two consecutive points by dividing the distance pen travels by the time difference.

3.2 Vertex detection

Stroke processing starts by looking for vertices. We use the sketch in Fig. 3-1 as a motivating example of what should be done in the vertex detection phase. Points marked in Fig. 3-1 indicate the corners of the stroke where the local curvature is high. Note that there are no vertices marked on the curved portion of the stroke. During the vertex detection process, we want to avoid picking points on the curved regions as much as possible. Piecewise linear approximation algorithms don't satisfy this requirement.

Vertex localization is a frequent subject in the extensive literature on graphics recognition (e.g., [16] compares 21 methods). Unfortunately these methods produce piecewise linear approximations. Our approach takes advantage of the interactive nature of sketching by combining information from both curvature and speed data for detecting corners while avoiding a piecewise linear approximation. For example,

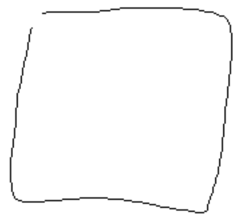


Figure 3-2: Stroke representing a square.

the direction, curvature and speed graphs for the square in figure 3-2 are in figure 3-3. We locate vertices by looking for points along the square where we have a local maximum in the absolute value of the curvature[1] or a minimum in the speed graph.

Although we said that the extrema in the curvature and speed data correspond to feature points in the original stroke, it is clear that we should not blindly compute the zero crossings of the derivative of the data, because the data is noisy. Doing so would introduce many false positives. In the following subsections we describe two methods that detect feature points in presence of noise, while avoiding false positives.

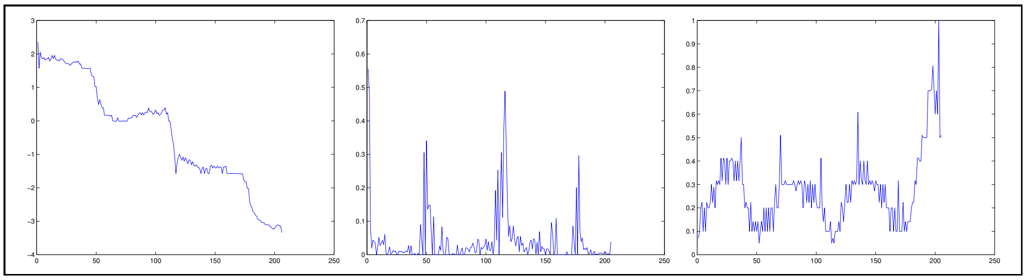


Figure 3-3: Direction, curvature and speed graphs for the stroke in figure 3-2

3.2.1 Average based filtering

False positives arise from looking at the extrema of the data at the most detailed scale. Extrema at the most detailed scale include those corresponding to the salient features of the data as well as those caused by the noise in the data. We want to find the extrema due to salient features, avoiding local extrema, while (of course) finding

[1] From this point on, we when we say curvature, we will refer to the absolute value of the curvature data.

more than just the single global extreme. To accomplish this, we select only the extrema of the function above a threshold. To avoid the problems posed by choosing a fixed threshold, we compute the threshold by computing the mean of the data and then scaling[2] it. We use this threshold to separate the data into regions where it is above/below the threshold. Then we select the global extrema within each region. We illustrate how this technique – average based filtering – can be applied to vertex detection using curvature and speed data.

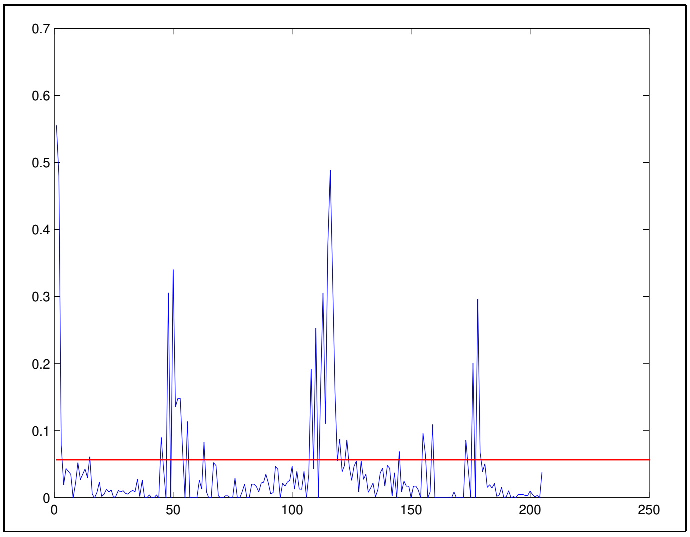


Figure 3-4: Curvature graph for the square in figure 3-2 with the threshold dividing it into regions.

3.2.2 Application to curvature data

Figure 3-4 shows how average based filtering partitions the curvature graph into different regions. Intuitively, the average based filtering partitions the stroke into regions of high and low curvature. Because we are interested in detecting the corners, we search for the maximum of the curvature within the regions with significant curvature. Note how this reduces, but doesn't eliminate, the problem of false positives introduced by noise in the captured stroke.

[2]This scaling factor is determined empirically. In our system we used the mean for curvature data, and scaled the mean by 0.9 for the speed data.

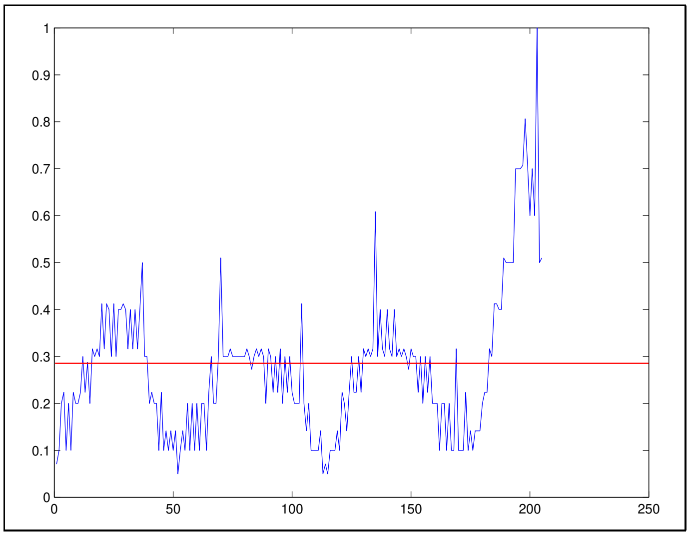


Figure 3-5: Speed graph for the stroke in figure 3-2 with the threshold dividing it into regions.

Although this average based filtering method performs better than simply comparing the curvature data against a hard coded threshold, it is not completely constant free. As we explain later, using the scale space provides a better methodology for dealing with noisy data without making a priori assumptions about the scale of relevant features.

3.2.3 Application to speed change

Our experience is that curvature data alone rarely provides sufficient reliability. Noise is one problem, but variety in angle changes is another. Fig. 3-6 illustrates how curvature alone fit misses a vertex (at the upper right) because the curvature around that point was too small to be detected in the context of the other, larger curvatures. We solve this problem by incorporating the speed data into our decision as an independent source of guidance.

Just as we did for the curvature data, we filter out the false extrema by average based filtering, then look for speed minima. The intuition here is simply that pen speed drops when going around a corner in the sketch. Fig. 3-7 shows (at left) the speed data for the sketch in Fig. 3-6, along with the polygon drawn from the speed-detected vertices (at right).

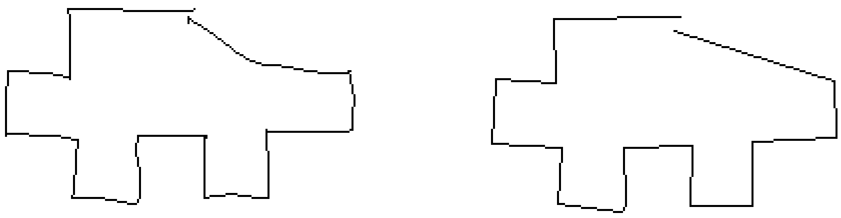


Figure 3-6: At left the original sketch of a piece of metal; at right the fit generated using only curvature data.

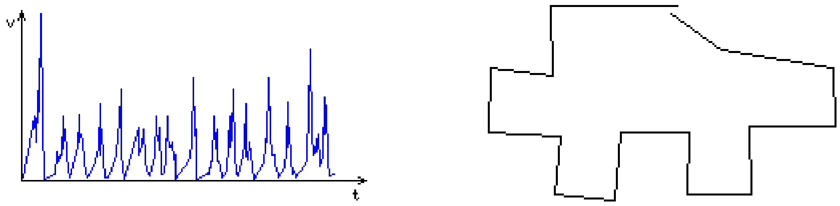


Figure 3-7: At left the speed graph for the piece; at right the fit based on only speed data.

3.3 Scale space approach

An inherent property of real-world objects is that they exist as meaningful entities over a range of scales. The classical example is a tree branch. A tree branch is meaningful at the centimeter or meter levels. It would be absurd to measure or look at it at very small scales where cells, molecules or atoms would make sense, or at a very large scale where it makes sense to talk about forests and trees rather than branches.

As humans we are good detecting features at multiple scales, but when we use computers to interpret sampled data, we have to take features at multiple scales into account, because digital data is degraded by noise and digitization.

In the case of stroke approximation, there are problems posed by noise and digitization. In addition, selecting an a priori scale has the problem of not lending itself to different scenarios where object features and noise may vary. There's a need to remove the dependence of our algorithms on preset thresholds.

A technique for dealing with features at multiple scales is to look at the data through multiple scales. The scale space representation framework introduced by Witkin [21] attempts to remove the dependence on constant thresholds and making a priori assumptions about the data. It provides us with a systematic framework for dealing with the kind of data we are interested in.

The virtues of scale-space approach are twofold. First it enables multiple interpretations of the data. These interpretations range from descriptions with high detail to descriptions that capture only the overall structure of the stroke. The second virtue of having representations of the data at multiple scales is setting the stage for selecting a scale or a set of scales by looking at how the interpretation of the data changes and features move in the scale space as the scale is varied.

The intuition behind scale-space representation is generating successively higher level descriptions of a signal by convolving it with a filter that does not introduce new feature points as the scale increases.

As a filter we use the Gaussian function, defined as:

$$g(s, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-s^2/2\sigma^2}$$

where σ is the smoothing parameter that controls the scale. Higher σ means coarser scales describing the overall features of the data, while a smaller σ corresponds to finer scales containing the details.

The Gaussian filter satisfies the restriction of not introducing new feature points. The uniqueness of the Gaussian kernel for use in scale-space filtering is discussed in [22] and [2].

Given a function $f(x)$, the convolution is given by:

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

We use the discrete counterpart of the Gaussian function which satisfies the property:

$$\sum_{i=0}^{n} g(i, \sigma) = 1$$

Given a Gaussian kernel, we convolve the data using the following scheme:

$$x_{(k,\sigma)} = \sum_{i=0}^{n} g(i, \sigma) x_{k - \lfloor n/2+1 \rfloor + i}$$

There are several methods for handling boundary conditions when the extent of the kernel is beyond end points. In our implementation we assume that for $k - \lfloor n/2 + 1 \rfloor + i < 0$ and $k - \lfloor n/2 + 1 \rfloor + i > n$ the data is padded with zeroes on either side.

In the pattern recognition community [5], [14] and [12] apply some of the ideas from scale space theory to similar problems. In particular [5], and [14] apply scale-space idea to detection of corners of planar curves and shape representation.

Scale space provides a concise representation of the behavior of the data across scales, but doesn't provide a generic scale selection methodology. There is no known task independent way of deciding which scales are important when looking at the scale-space map for some data. On the other hand it is possible to formulate scale selection methods by observing the properties of the scale-space map for a given task such as edge detection or ridge detection. In the following subsections, we explain how we used the feature count for scale selection in shape approximation. Our goal is selecting a scale where the extrema due to noise disappear.

3.3.1 Application to curvature data

As we did in the average based filtering, we start by deriving direction and curvature data. Next we derive a series of functions from the curvature data by smoothing it with Gaussian filters of increasing σ. Then we find the zero crossings of the curvature at each scale and build the scale-space.

Scale-space is the (x, σ)-plane where x is the dependent variable of function $f(.)$ [21]. We focus on how maxima of curvature move in this 2D plane as σ is varied.

Fig 3-8 shows a freehand stroke and Fig. 3-9 the scale space map corresponding to the features obtained using curvature data. The vertical axis in the graph is the scale index σ (increasing up). The horizontal axis ranging from 0 to 178 indicates the indices of the feature points in the scale space. The stroke in question contains

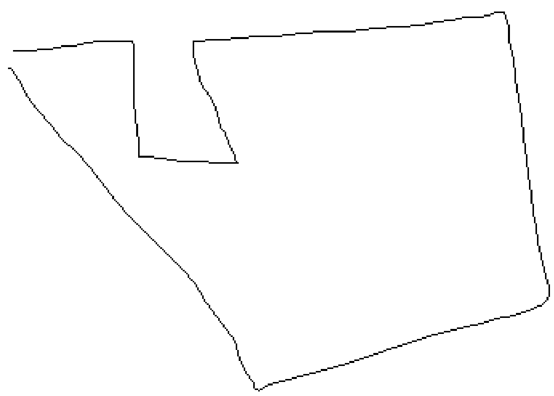


Figure 3-8: A freehand stroke.

179 points. We detect the feature points by finding the negative zero-crossings of the derivative of absolute value of the curvature at a particular scale. We do this at each scale and plot the corresponding point (σ, i) for each index i in the scale space plot. An easy way of reading this plot is by drawing a horizontal line at a particular scale index, and then looking at the intersection of the line with the scale-space lines. The intersections indicate the indices of the feature points at that scale.

As seen in this graph, for small σ (bottom of the scale space graph), many points in the stroke end up being detected as vertices because at these scales the curvature data has many local maxima, most of which are caused by the noise in the signal. For increasing σ, the number of feature points decreases gradually, and for the largest scale σ_{max} (top of the scale space graph), we have only three feature points left, excluding the end points.

Our goal at this stage is to choose a scale where the false positives due to noise are filtered out and we are left with the real vertices of the data. We want to achieve this without having any particular knowledge about the noise[3] and without having preset scales or constants for handling noise.

The approach we take is to keep track of the number of feature points as a function of σ and find a scale preserving the tradeoff between choosing a fine scale where the data is too noisy and introduces many false positives, and choosing a coarse scale

[3]The only assumption we make is that the noise is smaller in magnitude than the feature size.

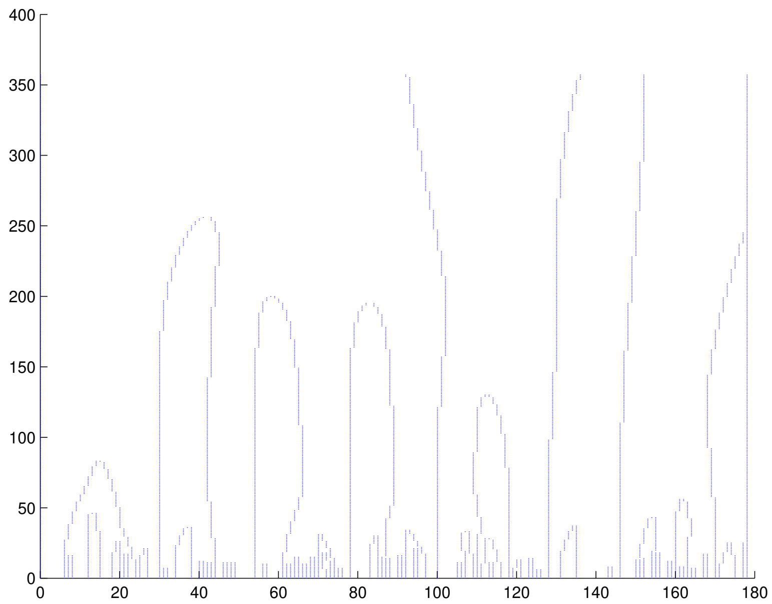


Figure 3-9: The scale-space for the maxima of the absolute curvature for the stroke in 3-8. This plot shows how the maxima move in the scale space. The x axis is the indices of the feature points, the y axis is the scale index.

where true feature points are filtered out. For example, the stroke in Fig. 3-8, has 101 feature points for $\sigma = 0$. On the coarsest scale, we are left with only 5 feature points, two of which are end points. This means 4 actual feature points are lost by the Gaussian smoothing. Because the noise in the data and the shape described by the true feature points are at different scales, it becomes possible to detect the corresponding ranges of scales by looking at the feature count graph.

For this stroke, the feature count graph is given in Fig. 3-10. In this figure, the steep drop in the number of feature points that occurs for scale indices $[0, 40]$ roughly corresponds to scales where the noise disappears, and the region $[85, 357]$ roughly corresponds to the region where the real feature points start disappearing. Fig. 3-11 shows the scale space behavior during this drop by combining the scale-space with the feature-count graph. In this graph, the x, y, axis z, respectively correspond to the feature point index [0,200], σ [0,400], and feature count [0,120]. We read the graph as follows: given σ, we find the corresponding location in the y axis. We move up parallel to the z axis until we cross the first scale space line[4]. The z value at which

[4]The first scale space line corresponds to the zeroth point in our stroke, and by default it is a

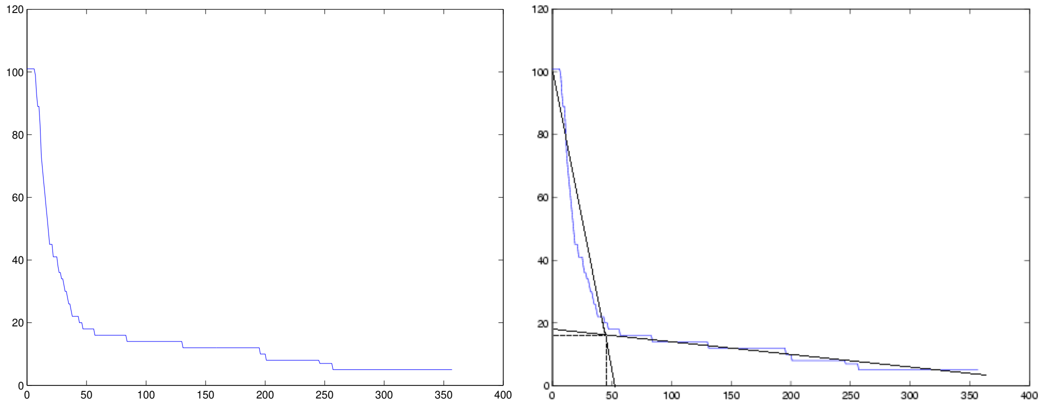


Figure 3-10: The plot on the left shows the drop in feature point count for increasing σ. The plot at right shows the scale selected by our algorithm (in both, the y axis is the feature count, x is the scale index).

we cross the first scale space line gives the feature count at scale index σ. Now, we draw an imaginary line parallel to the x axis. Movements along this line correspond to different feature indices, and its intersection with the scale space plot corresponds to indices of feature points present at scale index σ. The steep drop in the feature count is seen in both Fig. 3-10 and Fig. 3-11.

Our experiments suggest that this phenomena (i.e., the drop) is present in all hand drawn curves. For scale selection, we make use of this observation. We model the feature count - scale graph by fitting two lines and derive the scale using their intersection. Specifically, we compute a piecewise linear approximation to the feature count - scale graph with only two lines, one of which tries to approximate the portion of the graph corresponding to the drop in the number of feature points due to noise, and the other that approximates the portion of the graph corresponding to the drop in the number of real feature points. We then find the intersection of these lines and use its x value (i.e., the scale index) as the scale. Thus we avoid extreme scales and choose a scale where most of the noise is filtered out.

Fig. 3-10 illustrates the scale selection scheme via fitting two lines l_1, l_2 to the feature count - scale graph. The algorithm to get the best fit simply finds i that minimizes $OD(l_1, \{P_j\}) + OD(l_2, \{P_k\})$ for $0 \leq j < i$, $i \leq k < n$. $OD(l, \{P_m\})$ is the

feature point and is plotted in the scale space plot. This remark also applies to the last point in the stroke.

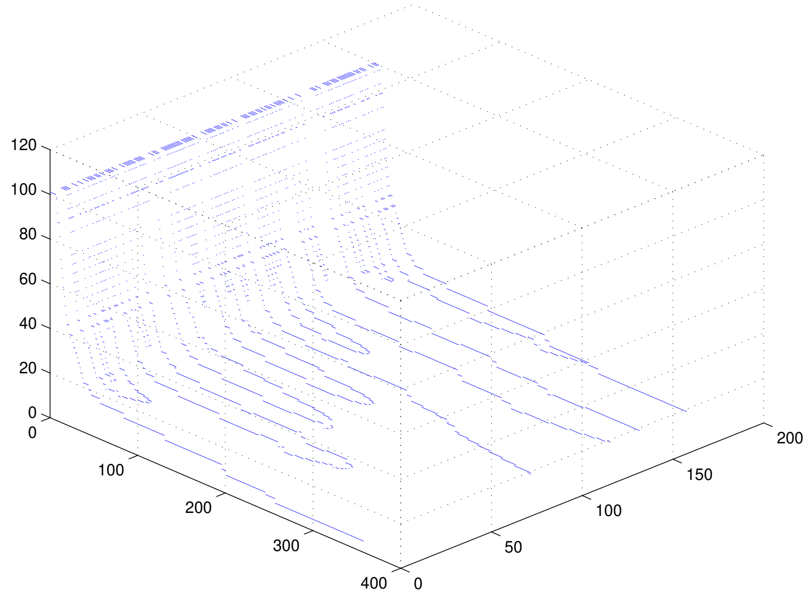


Figure 3-11: Joint scale-space feature-count graph for the stroke in Fig. 3-8. This plot simultaneously shows the movement of feature points in the scale space and the drop in feature point count for increasing σ. Here the z axis is the feature count [0,120], the x axis is the feature point index [0,200], and the y axis is the scale index [0,400].

average orthogonal distance of the points P_m to the line l, P is the array of points in the feature count - scale graph indexed by the scale parameter and $0 \leq i < n$ where n is the number of points in the stroke. Intuitively, we divide the feature count - scale graph into two regions, fit an ODR line to each region, and compute the orthogonal least squares error for each fit. We search for the division that minimizes the sum of these errors, and select the scale corresponding to the intersection of the lines for which the division is optimal (i.e., has minimum error).

Interestingly enough, we have reduced the problem of stroke approximation via feature detection to fitting lines to the feature count graph, which is similar in nature to the original problem. However, now we know how we want to approximate the data (i.e., with two lines). Therefore even an exhaustive search for i corresponding to the best fit becomes feasible. As shown in Fig. 3-12 the error as a function of i is a U shaped function. Thus, if desired, the minima of the summed error can be found using gradient descent methods by paying special attention to not getting stuck in the local minima. For the stroke in Fig. 3-8, the scale index selected by our algorithm

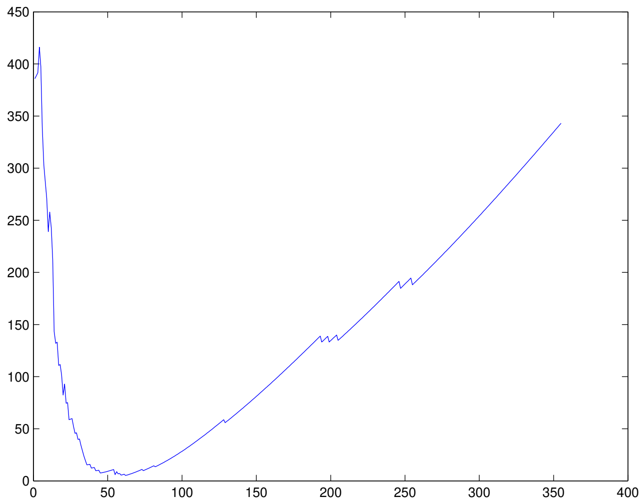


Figure 3-12: The summed error for the two lines fit to Fig. 3-10 during scale selection for the stroke in Fig. 3-8.

is 47.

While we try to choose a scale where most of the false maxima due to noise are filtered out, feature points at this scale we may still contain some false positives. The problem of false extrema in the scale space is also mentioned in [14], where these points are filtered out by looking at their separation from the line connecting the preceding and following feature points. They filter these points out if the distance is less than one pixel.

The drawback of this filtering technique is that the scale-space has to be built differently. Instead of computing the curvature for $\sigma = 0$ and then convolving it with Gaussian filters of larger σ to obtain the curvature data at a particular scale, they treat the stroke as a parametric function of a third variable s, path length along the curve. The x and y components are expressed as parametric functions of s. At each scale, the x and y coordinates are convolved with the appropriate Gaussian filter and the curvature data is computed. It is only after this step that the zero crossings of the derivative of curvature can be computed for detecting feature points. The x and y components should be convolved separately because filtering out false feature points requires computing the distance of each feature point to the line connecting the preceding and following feature points, as explained above. This means the Gaussian

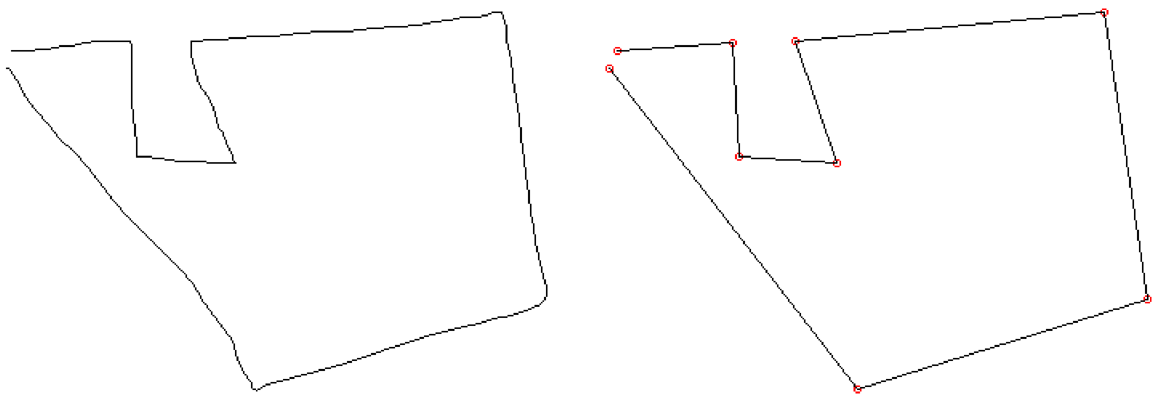


Figure 3-13: The input stroke (on the left) and the features detected by looking at the scale space of the curvature (on the right).

convolution, a costly operation, has to be performed twice in this method, compared to a single pass in our algorithm.

Because we convolve the curvature data instead of the x and y coordinates, we can't use the method mentioned above. Instead we use an alternate 2-step method to remove the false positives. First we check whether there are any vertices that can be removed without increasing the least squares error between the generated fit and the original stroke points[5]. The second step in our method takes the generated fit, detects consecutive collinear[6] edges and combines these edges into one by removing the vertex in between. After performing these operations, we get the fit in Fig. 3-13.

One virtue of the scale space approach is that works extremely well in the presence of noise. In Fig.3-14 we have a very noisy stroke. Figures 3-15 and 3-16 show the feature-count and scale-space respectively. Fig.3-17 combines these two graphs, making it easier to see simultaneously what the feature count is at a particular scale, and what the scale-space behavior is in that neighborhood.

The output of the scale-space based algorithm is in Fig. 3-18. This output contains only 9 points. For comparison purposes, the output of the average based feature detection algorithm based on curvature is also given in Fig. 3-18. This fit contains 69

[5]It is also possible to relax this criteria and remove points if the increase in the least squares error of the segment they belong to remains within some percentage of the original error.

[6]The collinearity measure is determined by the task in hand. In our system, lines intersecting with an angle of $\pi/32$ or less are considered to be collinear.

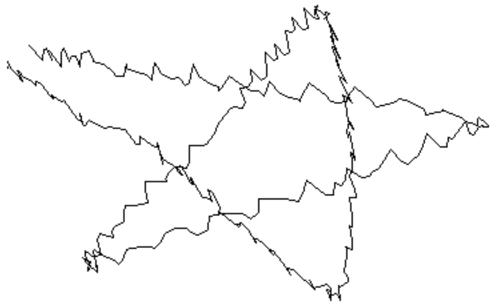


Figure 3-14: A very noisy stroke.

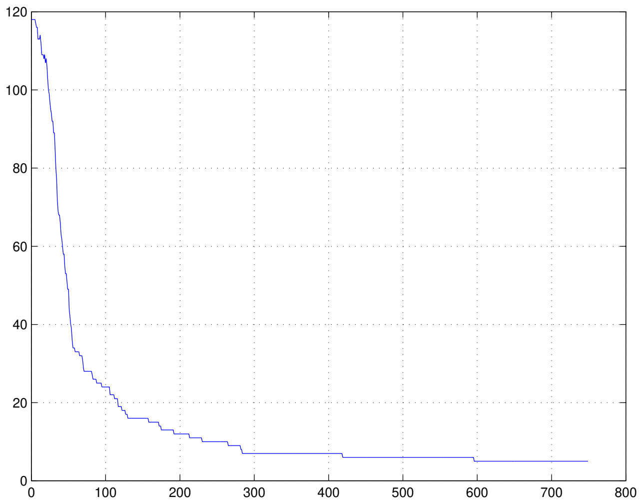


Figure 3-15: This plot shows the drop in feature point count for increasing σ. y axis is the feature count, and the x axis is the scale index. Even in the presence of high noise, the behavior in the drop is the same as it was for 3-8. Fig.3-17 combines this graph with the feature count graph to illustrate the drop in the feature count and the scale-space behavior.

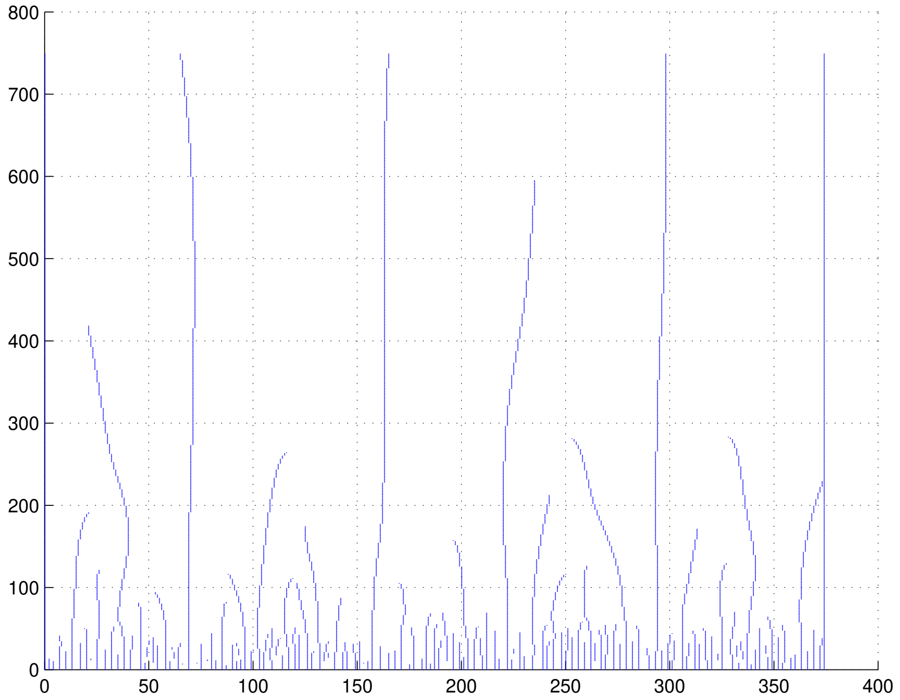


Figure 3-16: The scale space map for the stroke in Fig. 3-14. Fig.3-17 combines this graph with the feature count graph to illustrate the drop in the feature count and the scale-space behavior.

vertices. (The vertices are not marked for the sake of keeping the figure clean.)

3.3.2 Application to speed data

We applied the scale selection technique mentioned above on speed data. The details of the algorithm for deriving the scale-space and extracting the feature points are similar to that of the curvature data, but there are some differences. For example, instead of looking for the maxima, we look for the minima.

Fig. 3-20 has the scale-space, feature-count and joint graphs for the speed data of the stroke in Fig. 3-14. As seen in these graphs, the behavior of the scale space is similar to the behavior we observed for the direction data. We use the same method for scale selection. In this case, the scale index picked by our algorithm was 72. The generated fit is in Fig. 3-19 along with the fit generated by the average based filtering method using the speed data.

For the speed data, the fit generated by scale-space method has 7 vertices, while

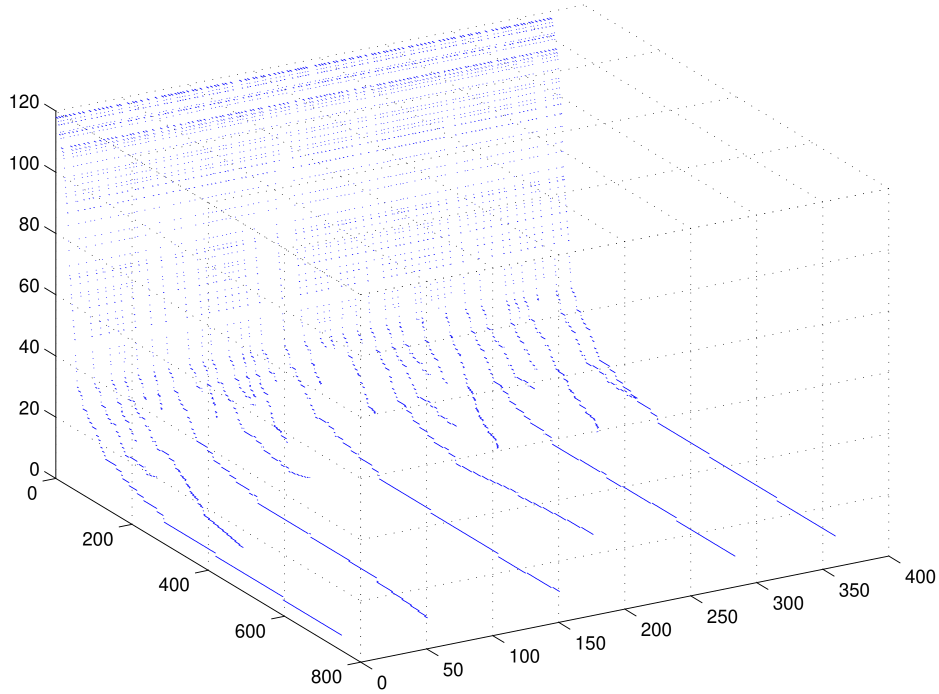


Figure 3-17: Joint feature-count scale-space graph obtained for the noisy stroke in Fig. 3-14 using curvature data. This plot simultaneously shows the movement of feature points in the scale space and the drop in feature point count for increasing σ. Here the z axis is the feature count [0,120], the x axis is the feature point index [0,400], and the y axis is the scale index [0,800].

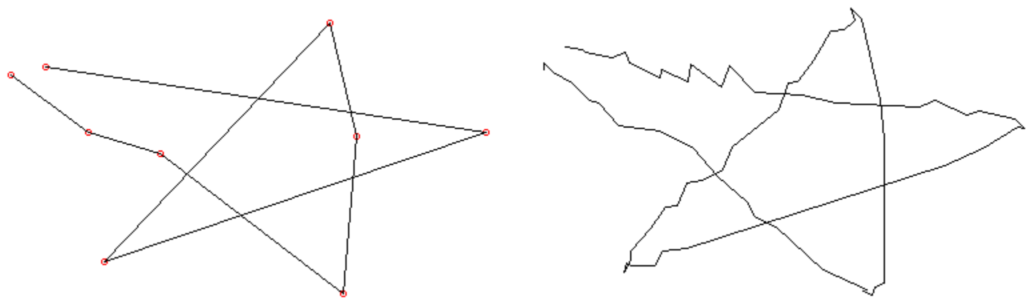


Figure 3-18: On the left is the fit obtained by the scale-space approach using curvature data for the stroke in Fig. 3-14. This fit has only 9 vertices. On the right, the fit generated by the average filtering with 69 features. The vertices are not marked to keep the figure uncluttered.

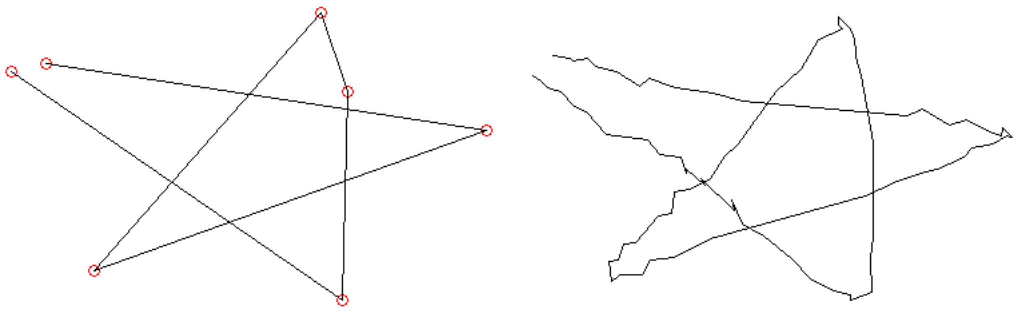


Figure 3-19: On the left, the fit generated by the scale-space approach using speed data (which has 7 vertices). On the right is the fit obtained by average filtering on speed data for the stroke in Fig. 3-14. This fit has 82 vertices that are not marked to keep it uncluttered.

the one generated by the average based filtering has 82. In general, the performance of the average based filtering method is not as bad as this example may suggest. For example, for strokes as in Fig.3-8, the performance of the two methods are comparable, but for extremely noisy data as in Fig. 3-14, the scale-space approach pays off. This remark also applies to the results obtained using curvature data. Because the scale-space approach is computationally more costly[7], using average based filtering is preferable for data that is less noisy. There are also scenarios where only one of curvature or speed data may be more noisy. For example, in some platforms, the system generated timing data for pen motion required to derive speed may not be precise enough, or may be noisy. In this case, if the noise in the pen location is not too noisy, one can use the faster average based method for generating fits from the curvature data and the scale-space method for deriving the speed fit. This is a choice that the user has to make based on the accuracy of hardware used to capture the strokes, and and the computational limitations.

We conclude by an example illustrating that our algorithm satisfies the requirement mentioned at the beginning of the chapter, namely that of avoiding piecewise linear approximations at curved portions of the input stroke. Fig. 3-21 contains a

[7]Computational complexity of the average based filtering is linear with the number of points where the scale space approach requires quadratic time if the scale index is chosen to be a function of the stroke length.

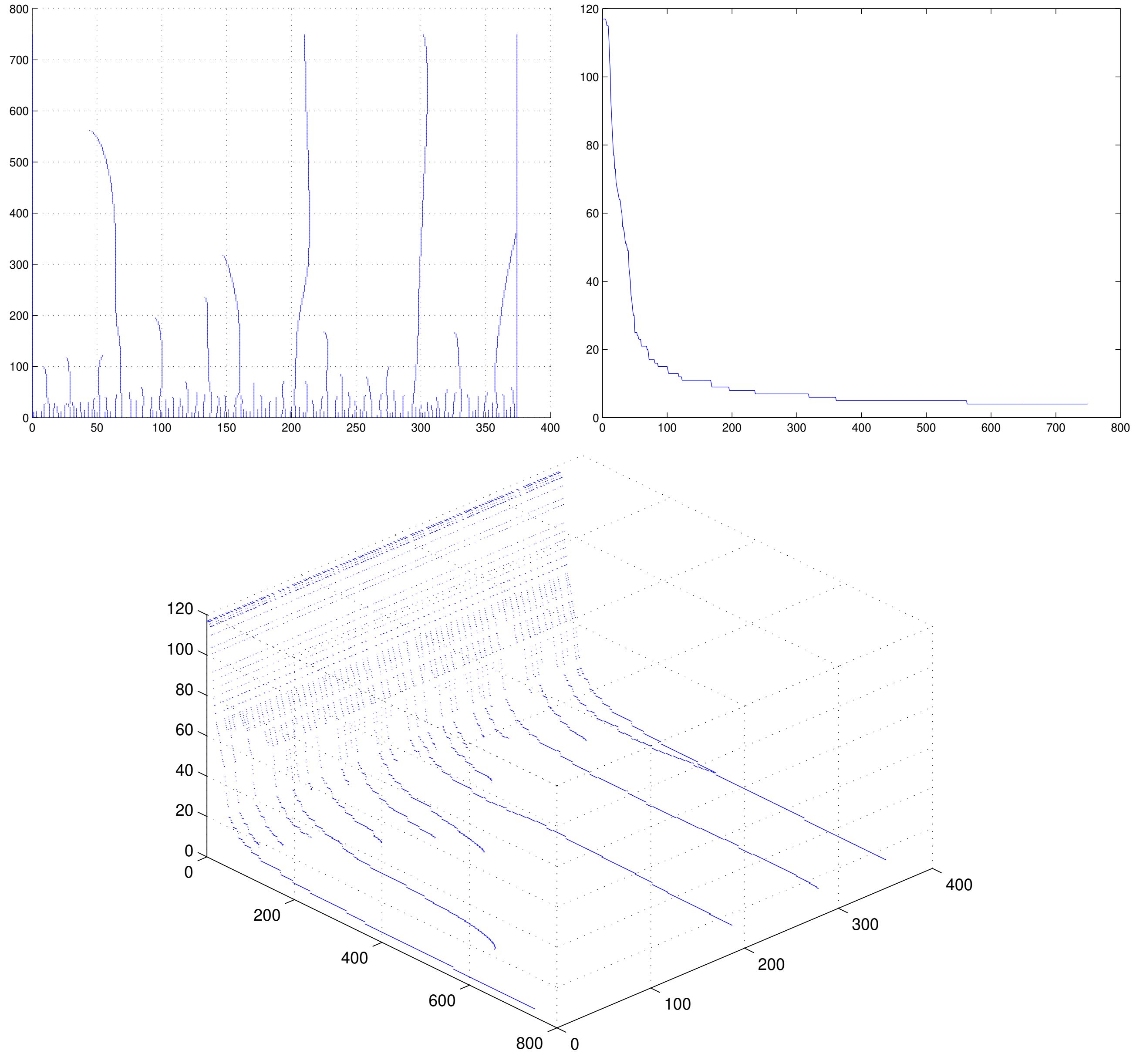


Figure 3-20: The scale-space, feature-count and joint graphs for the speed data of the stroke in Fig.3-14. In this case, the scale selected by our algorithm is 72.

stroke consisting of curved and straight segments and the features points picked by the scale-space based algorithm we described. Note that although some of the feature points are on the curve, the piecewise linear approximation described by these points is, by criteria we describe below, such a crude approximation to the original stroke that the system will fit Bézier curves to that segment (see section 5.1). In the future work chapter, we propose two possible directions that can be taken to remove these false positives if needed.

In this chapter, we have described several methods for detecting feature points of hand-drawn strokes. Next we describe how features detected by different methods can be combined for better performance.

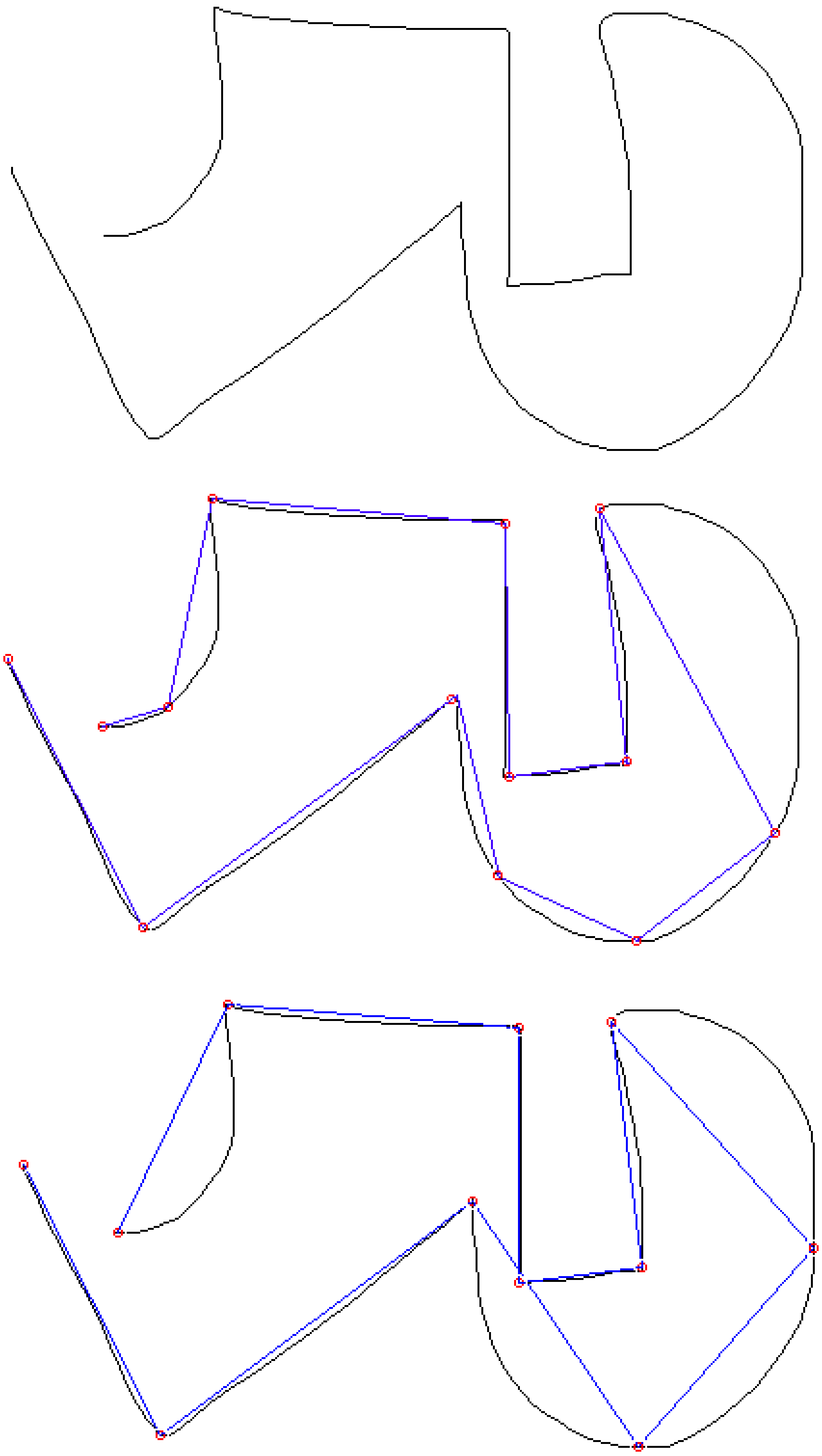


Figure 3-21: A stroke consisting of curved and straight segments and the feature points detected by the scale-space based algorithms we have described. The figure in the middle shows the feature points generated using curvature data and the one in the bottom using the speed data.

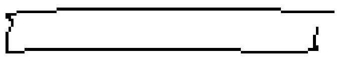
Chapter 4

Generating Hybrid Fits

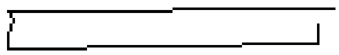
In the previous chapter we introduced methods for vertex detection using curvature and speed data. As we pointed out, curvature data itself was not sufficient to detect all vertices, motivating our use of speed data. However, using speed data alone has its shortcomings as well. Polylines formed by a combination of very short and long line segments can be problematic: the maximum speed reached along the short segments may not be high enough to indicate the pen has started traversing another edge, causing the entire short segment to be interpreted as a corner. This problem arises frequently when drawing thin rectangles, common in sketches of mechanical devices. Fig. 4-1 illustrates this phenomena. In this figure, the speed fit misses the upper left corner of the rectangle because the pen failed to gain enough speed between the endpoints of the corresponding short segment. Fig. 4-2 shows pen speed for this rectangle. The curvature fit, by contrast, detects all corners, along with some other vertices that are artifacts due to hand dynamics during freehand sketching.

Since, both curvature and speed data alone are insufficient for generating good fits in certain scenarios, a method to combine these two information sources is needed. We use information from both sources, and generate hybrid fits by combining the candidate set F_c obtained using curvature data with the candidate set F_s obtained using speed information, taking into account the system's certainty that each candidate is a real vertex.

Hybrid fit generation occurs in three stages: computing vertex certainties, gener-



(a) Input, 63 points



(b) Using curvature data, 7 vertices

(c) Using speed data, 4 vertices

Figure 4-1: Average based filtering using speed data misses a vertex. The curvature fit detects the missed point (along with vertices corresponding to the artifact along the short edge of the rectangle on the left).

ating a set of hybrid fits, and selecting the best fit.

4.1 Computing vertex certainties

Our certainty metric for a curvature candidate vertex v_i is the scaled magnitude of the curvature in a local neighborhood around it expressed by $|d_{i-k} - d_{i+k}|/l$. Here l is the curve length between points S_{i-k}, S_{i+k} and k is a small integer defining the neighborhood size around v_i. The certainty values are normalized to be within $[0, 1]$. The certainty metric for a speed fit candidate vertex v_i is a measure of the pen slowdown at the point, $1 - v_i/v_{max}$, where v_{max} is the maximum pen speed anywhere in the vertices of the approximation.

As is traditional both of these metrics produce values in $[0, 1]$, though with different scales. Metrics are used only for ordering within each set, so they need not be numerically comparable across sets. Candidate vertices are sorted by certainty within each fit.

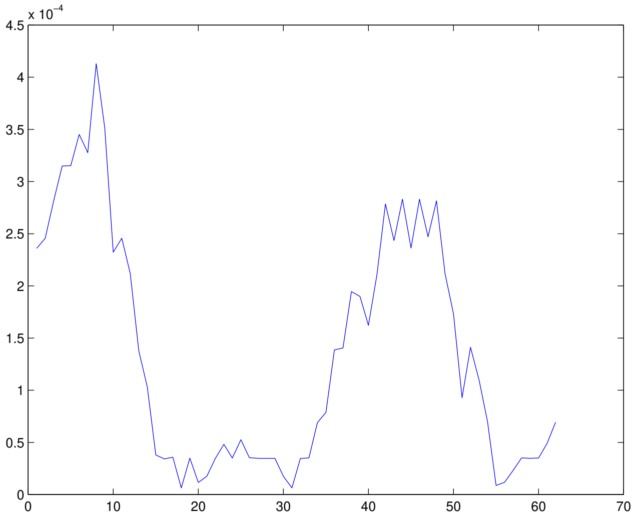


Figure 4-2: The speed data for the rectangle in Fig. 4-1.

4.2 Computing a set of hybrid fits

The initial hybrid fit H_0 is the intersection[1] of F_c and F_s. A succession of additional fits are then generated by appending to H_i the highest scoring curvature and speed candidates not already in H_i.

To do this, on each cycle we create two new fits: $H_i' = H_i \cup \{v_s\}$ (i.e., H_i augmented with the best remaining speed fit candidate) and $H_i'' = H_i \cup \{v_d\}$ (i.e., H_i augmented with the best remaining curvature candidate). We use least squares error as a metric of the goodness of a fit. The error ε_i is computed as the average of the sum of the squares of the distances to the fit from each point in the stroke S:

$$\varepsilon_i = \frac{1}{|S|} \sum_{s \in S} ODSQ(s, H_i)$$

Here $ODSQ$ stands for *orthogonal distance squared*, i.e., the square of the distance from the stroke point to the relevant line segment of the polyline defined by H_i. We compute the error for H_i' and for H_i''; the higher scoring of these two (i.e., the one with smaller least squares error) becomes H_{i+1}, the next fit in the succession. This

[1] The indices picked by the speed and curvature fits for a particular corner may be off by a small offset (one or two), and this is taken into consideration when comparing vertices in F_c and F_s so that correspondence between the fits can be correctly identified.

	Vertices	ε_i
H_0	4	47.247
H_1	5	1.031
H_2	6	0.846
H_3	7	0.844

Table 4.1: The vertex count and least squares error of the hybrid fits generated for the rectangle in Fig. 4-1.

process continues until all points in the speed and curvature fits have been used. The result is a set of hybrid fits. Table 4.1 shows number of vertices for each H_i and the least squares errors for the thin rectangle in Fig 4-1. As expected, the errors decrease as the number of vertices increases.

4.3 Selecting the best of the hybrid fits

In selecting the best of the hybrid fits the problem is as usual trading off more vertices in the fit against lower error. Here our approach is simple: We set an error upper bound and designate as our final fit H_f, the H_i with the fewest vertices that also has an error below the threshold.

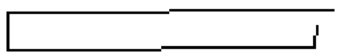
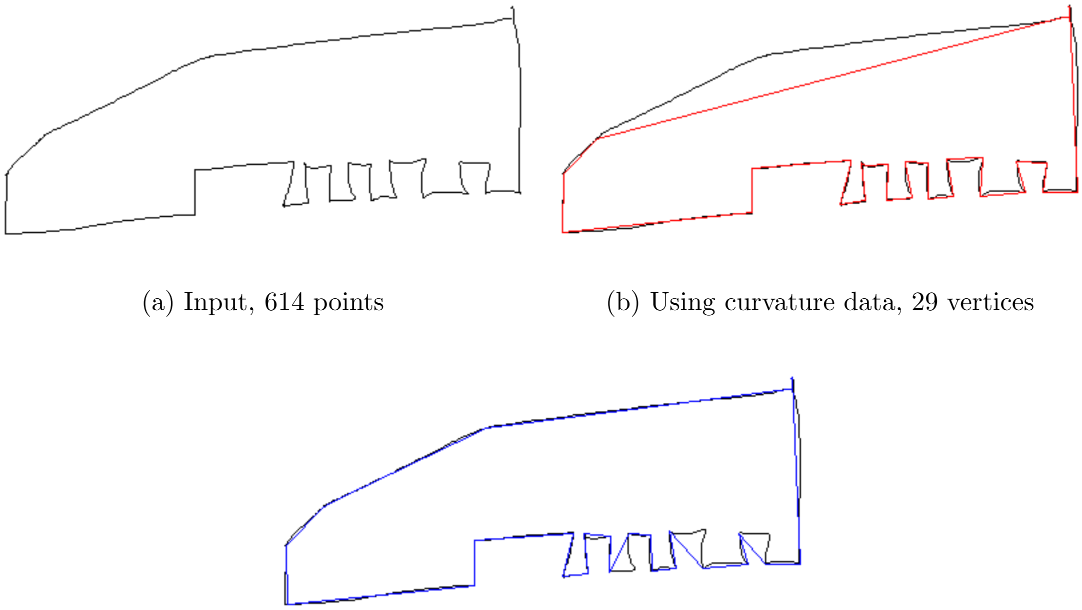


Figure 4-3: The hybrid fit chosen by our algorithm, containing 5 vertices.

The example above is a simple one where we miss one vertex using the speed information, and the curvature data detects all the vertices. Our method works even if neither the speed nor the curvature fits capture all the true vertices. Our algorithm handles such cases successfully.

The stroke in Fig. 4-4 was deliberately[2] drawn to include two soft corners along with 23 sharp corners making it hard to detect the soft corners using curvature data. It also includes many short edges that are particularly hard to detect using the speed

[2]It took the author several minutes to generate an example where both methods miss vertices.



(a) Input, 614 points

(b) Using curvature data, 29 vertices

(c) Using speed data, 24 vertices

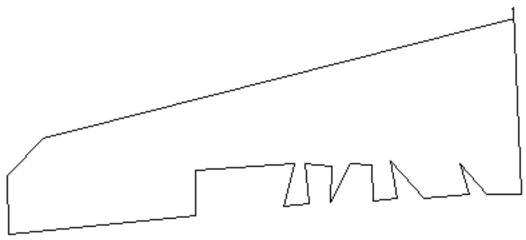
Figure 4-4: An example where the curvature data along with average based filtering misses points. The feature points are detected using the average based filtering. As seen here, the curvature fit missed some of the smoother corners (the fits generated by each method are overlayed on top of the original stroke to indicate missed vertices).

data alone. The fits generated by the curvature and speed data are in the same figure, and they are drawn over the original stroke to emphasize the vertices missed by each method.

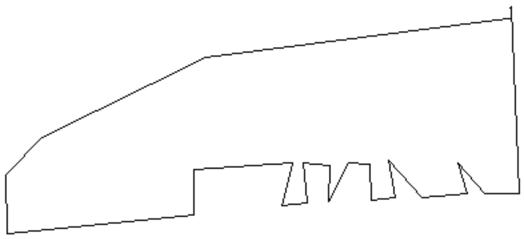
The hybrid fits generated by our method are in Fig. 4-5. In this case the hybrid fit chosen by our algorithm is H_4, containing all the corners. A summary of the relevant information for each hybrid fit is in table 4.2.

When average based filtering is used, in order for the hybrid fit generation to fail, the input stroke should satisfy the following criteria:

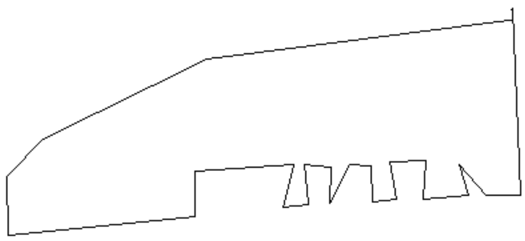
- The stroke should contain very short segments with shallow turns and smooth corners so that the curvature fit misses the corresponding vertices. In addition, these segments should be drawn slowly, so that vertices are missed by the speed fit as well.



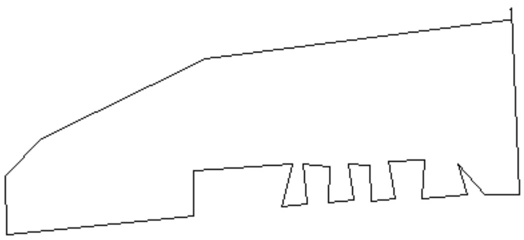
(a) $\varepsilon_0 = 89.8576$, 23 vertices



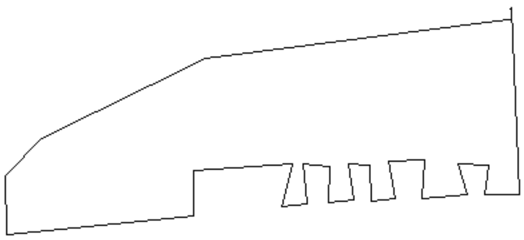
(b) $\varepsilon_1 = 14.9301$, 24 vertices



(c) $\varepsilon_2 = 8.9211$, 25 vertices



(d) $\varepsilon_3 = 5.6916$, 26 vertices



(e) $\varepsilon_4 = 1.4458$, 27 vertices

Figure 4-5: The series of hybrid fits generated for the complex stroke in Fig. 4-4. The fits successively get better.

- There should be many sharp turns in the shape to pull the average curvature up so that the vertices with the shallow turns are missed in such a context.

- The shape should contain long segments drawn rather fast, so the speed threshold is pulled up (causing the vertices to be missed).

Obviously it is very hard to satisfy all of these requirements.

Fig. 4-6 is our attempt to generate a shape that tries to satisfy the above requirements. Note how the region intended to have shallow turns starts looking like a

	Vertices	ε_i
H_0	23	89.857
H_1	24	14.930
H_2	25	8.921
H_3	26	5.691
H_4	27	1.445

Table 4.2: The vertex count and least squares errors of the hybrid fits generated for the stroke in Fig. 4-4.

smooth curve even with a little smoothing [3]. We tested our system with this example tailored to break curvature and speed based methods for the same vertices, paying special attention to making the corners in the region of interest just smooth enough so the whole region doesn't look like a curve, thus our expectation from the algorithm remains reasonable. Fig. 4-7 shows one of our attempts to produce such a scenario. After numerous tries, we were unable to generate a case where the generated hybrid fit missed the vertices in question.

[3]Obviously there is a limit on the smoothness of the corners and the length of the edges in this region, because as the edges are made shorter and the corners become smoother, the shape turns into a curve.

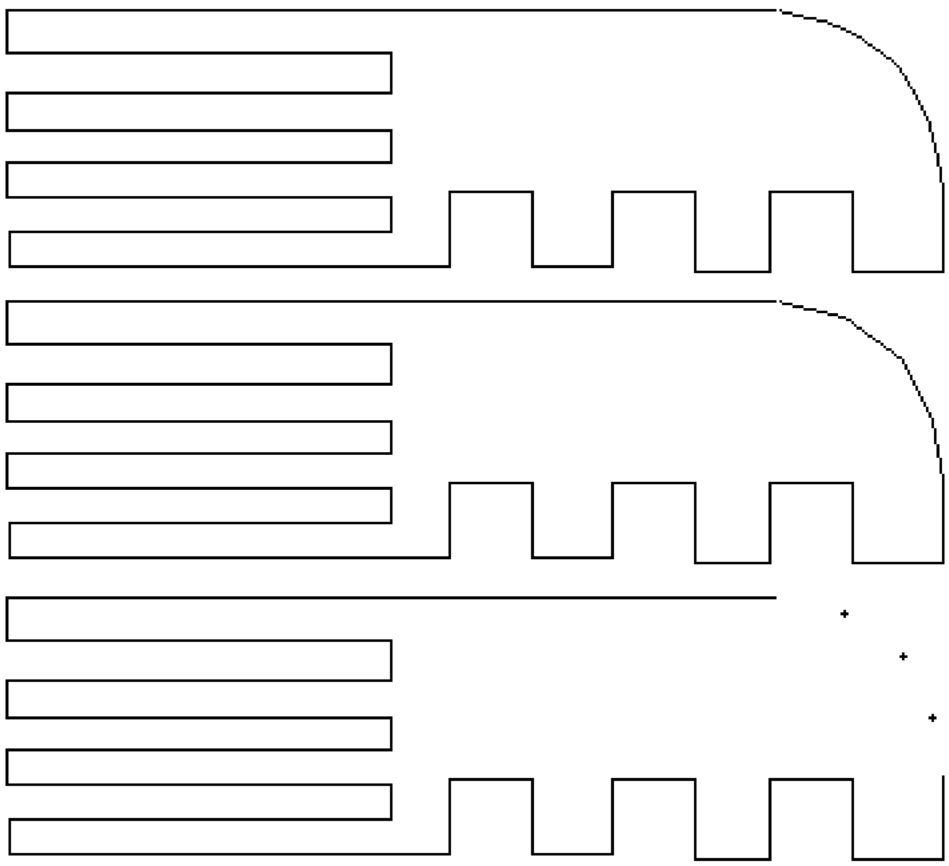


Figure 4-6: A shape devised to break curvature and speed based methods at the same time. The figure in the bottom shows the positions of the vertices intended by the user. The one in the middle combines these vertices with straight lines. The figure on top illustrates what the shape would look like if the corners in this region were made smoother.

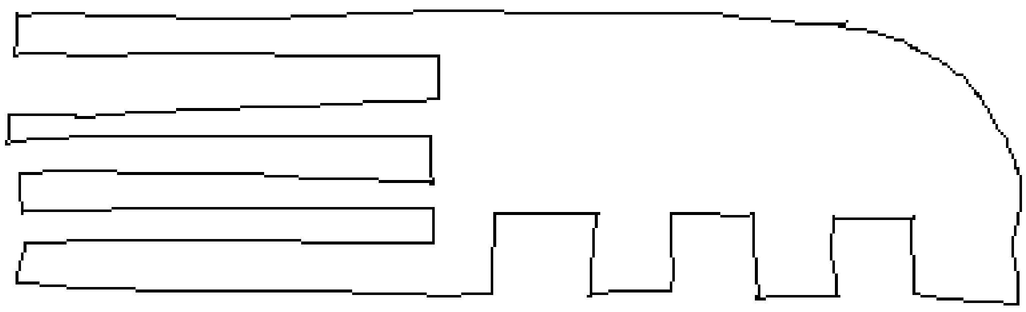


Figure 4-7: A stroke illustrating the kinds of strokes we drew trying to produce a shape as in Fig. 4-6.

Chapter 5

Handling Curves

The approach described so far yields a good approximation to strokes that consist solely of line segments, but as noted our input may include curves as well, hence we require a means of detecting and approximating them.

5.1 Curve detection

The feature points in the polyline approximation H_f generated at the end of hybrid fit selection process provide a natural foundation for detecting areas of curvature. We detect areas of curvature by comparing the Euclidean distance l_1 between each pair of consecutive vertices u, v in H_f to the accumulated arc length l_2 between the corresponding vertices in the input S. The ratio l_2/l_1 is very close to 1 in the linear regions of S, and significantly higher than 1 in curved regions.

5.2 Approximation

We detect curved regions by looking at the ratio l_2/l_1, and approximate these regions with Bézier curves, defined by two end points and two control points. Let $u = S_i$, $v = S_j$, $i < j$ be the end points of the part of S to be approximated with a curve. We compute the control points as:

$$c_1 = k\hat{t_1} + v$$

$$c_2 = k\hat{t_2} + u$$

$$k = \frac{1}{3} \sum_{i \le k < j} |S_k - S_{k+1}|$$

where $\hat{t_1}$ and $\hat{t_2}$ are the unit length tangent vectors pointing inwards at the curve segment to be approximated. The $1/3$ factor in k controls how much we scale $\hat{t_1}$ and $\hat{t_2}$ in order to reach the control points; the summation is simply the length of the chord between S_i and S_j.[1]

As in fitting polylines, we want to use least squares to evaluate the goodness of a fit, but computing orthogonal distances from each S_i in the input stroke to the Bézier curve segments would require solving a fifth degree polynomial. (Bézier curves are described by third degree polynomials, hence computing the minimum distance from an arbitrary point to the curve involves minimizing a sixth degree polynomial, equivalent to solving a fifth degree polynomial.) A numerical solution is both computationally expensive and heavily dependent on the goodness of the initial guesses for roots [15], hence we resort to an approximation. We discretize the Bézier curve using a piecewise linear curve and compute the error for that curve. This error computation is $O(n)$ because we impose a finite upper bound on the number of segments used in the piecewise approximation.

If the error for the Bézier approximation is higher than our maximum error tolerance, the curve is recursively subdivided in the middle, where middle is defined as the data point in the original stroke whose index is midway between the indices of the two endpoints of the original Bézier curve. New control points are computed for each half of the curve, and the process continues until the desired precision is achieved.

The capability of our approach is shown in figures 5-1 and 5-2 by a number of

[1]The $1/3$ constant was determined empirically, but works very well for freehand sketches. As we discovered subsequently, the same constant was independently chosen in [18].

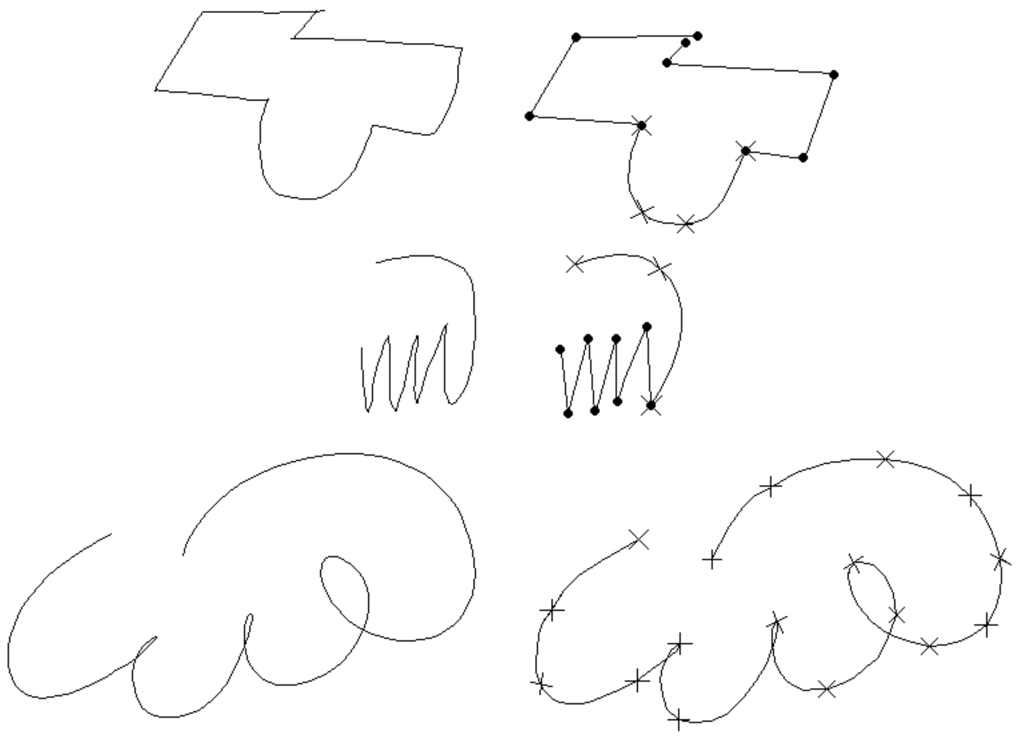


Figure 5-1: Examples of stroke approximation. Boundaries of Bézier curves are indicated with crosses, vertices are indicated with dots.

hastily-sketched strokes consisting of mixture of lines and curves.

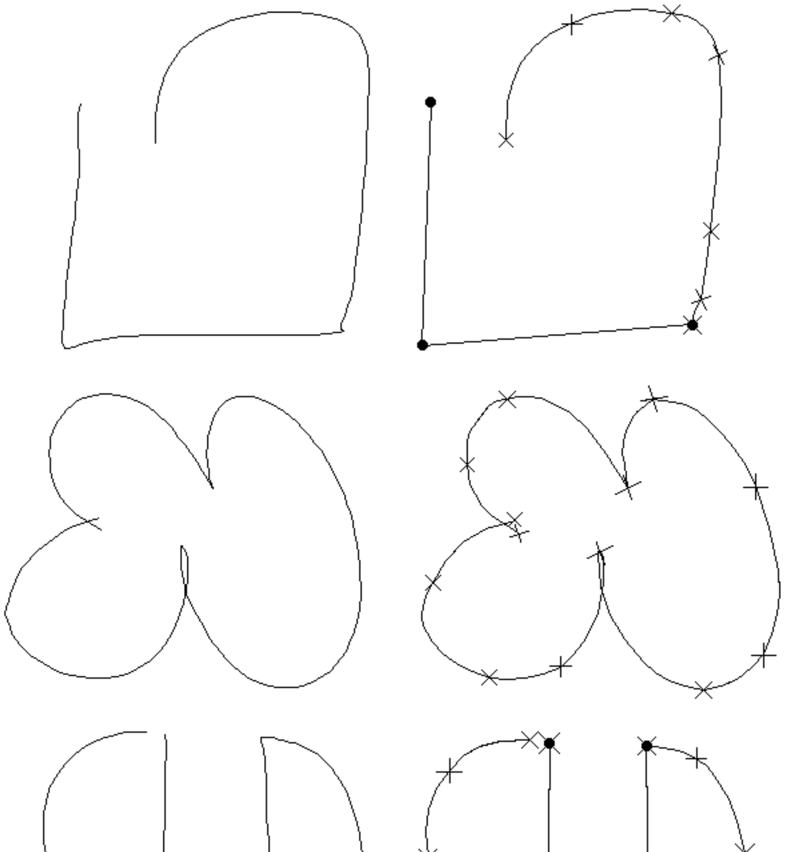


Figure 5-2: More examples of stroke approximation.

Chapter 6

Recognition

So far, we have described a system that generates a concise description of digital ink in terms of geometric primitives such as lines, ovals, polylines, curves, and their combination. In a sketch understanding system, such concise descriptions are more useful than descriptions in terms of pixel positions traced by the pen. Now we describe how the output of the approximation layer is further processed by beautification and recognition layers that exploit parallelism and achieve higher level, domain specific recognition respectively.

6.1 Beautification

Beautification refers to the adjustments made to the approximation layer's output, primarily to make it look as intended. We adjust the slopes of the line segments in order to ensure the lines that were apparently meant to have the same slope end up being parallel. This is accomplished by looking for clusters of slopes in the final fit produced by the approximation phase, using a simple sliding-window histogram. Each line in a detected cluster is then rotated around its midpoint to make its slope be the weighted average of the slopes in that cluster.

The (new) endpoints of these line segments are determined by the intersections of each consecutive pair of lines. This process may result in vertices being moved. We chose to rotate the edges about their midpoints because this produces vertex locations

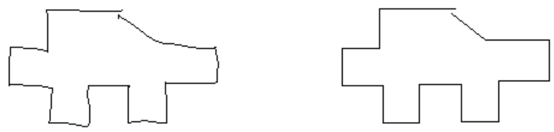


Figure 6-1: At left the original sketch of a piece of metal revisited, and the final beautified output at right.

that are close to those detected, have small least square errors when measured against the original sketch, and look right to the user. The movement of vertices as a result of beautification is unavoidable, because requiring the vertices to remain fixed results in an over-constrained system.

Fig. 6-1 shows the original stroke for the metal piece we had before, and the output of the beautifier. Some examples of beautification are also present in Fig. 6-3.

6.2 Basic Object Recognition

The next step in our processing is recognition of the most basic objects that can be built from the line segments and curve segments produced thus far, i.e., simple geometric objects (ovals, circles, rectangles, squares).

Recognition of these objects is done with hand-tailored templates that examine various simple properties. A rectangle, for example, is recognized as a polyline with 4 segments, all of whose vertices are within a specified distance of the center of the figure's bounding box. A stroke will be recognized as an oval if it has a small least squares error when compared to an oval whose axes are given by the bounding box of the stroke.

6.3 Evaluation

In order to evaluate our system, we wrote a higher level recognizer that takes the geometric descriptions generated by the basic object recognition and combines them into domain specific objects.

Higher level recognition is a difficult task. For example, in [10] a recognition rate of 63% is noted for the user interface sketching task despite the restricted set of low level primitives considered (in this case, rectangles, circles, lines, and squiggly lines for text). There is no doubt that in settings where larger sets of primitives are allowed, the performance will deteriorate simply because the low level recognition will make more errors. In this thesis, our focus is on the low level recognition (i.e., stroke approximation), and the recognition capability described in this chapter is implemented mainly for an informal evaluation of the stroke approximation layer as we demonstrate below.

Fig. 6-3 shows the original input and the program's analysis for a variety of simple but realistic mechanical devices drawn as freehand sketches. The last two of them are different sketches for a part of the direction reversing mechanism for a tape player. These examples also show some higher level domain specific recognition. Recognized domain specific components include gears (indicated by a circle with a cross), springs (indicated by wavy lines), and the standard fixed-frame symbol (a collection of short parallel lines). Components that are recognized are replaced with standard icons scaled to fit the sketch.

At this point the only evaluation is an informal comparison of the raw sketch and the system's approximations, determining whether the system has selected vertices where they were drawn, fit lines and curves accurately, and successfully recognized basic geometric objects. While informal, this is an appropriate evaluation because the program's goal is to produce an analysis of the strokes that "looks like" what was sketched.

We have also begun to deal with overtracing, one of the (many) things that distinguishes freehand sketches from careful diagrams. Fig. 6-2 illustrates one example of the limited ability we have thus far embodied in the program. We have observed that users overtrace more often when drawing ovals and lines. We detect overtraced lines by looking at the aspect ratio of the stroke's bounding box with its length. Overtraced ovals are handled by the low level recognition method described previously. The ability to handle overtracing is rather limited in the current system and a more formal approach is needed to handle overtracing in general.

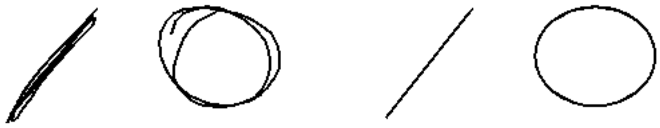


Figure 6-2: An overtraced oval and a line along with and the system's output.

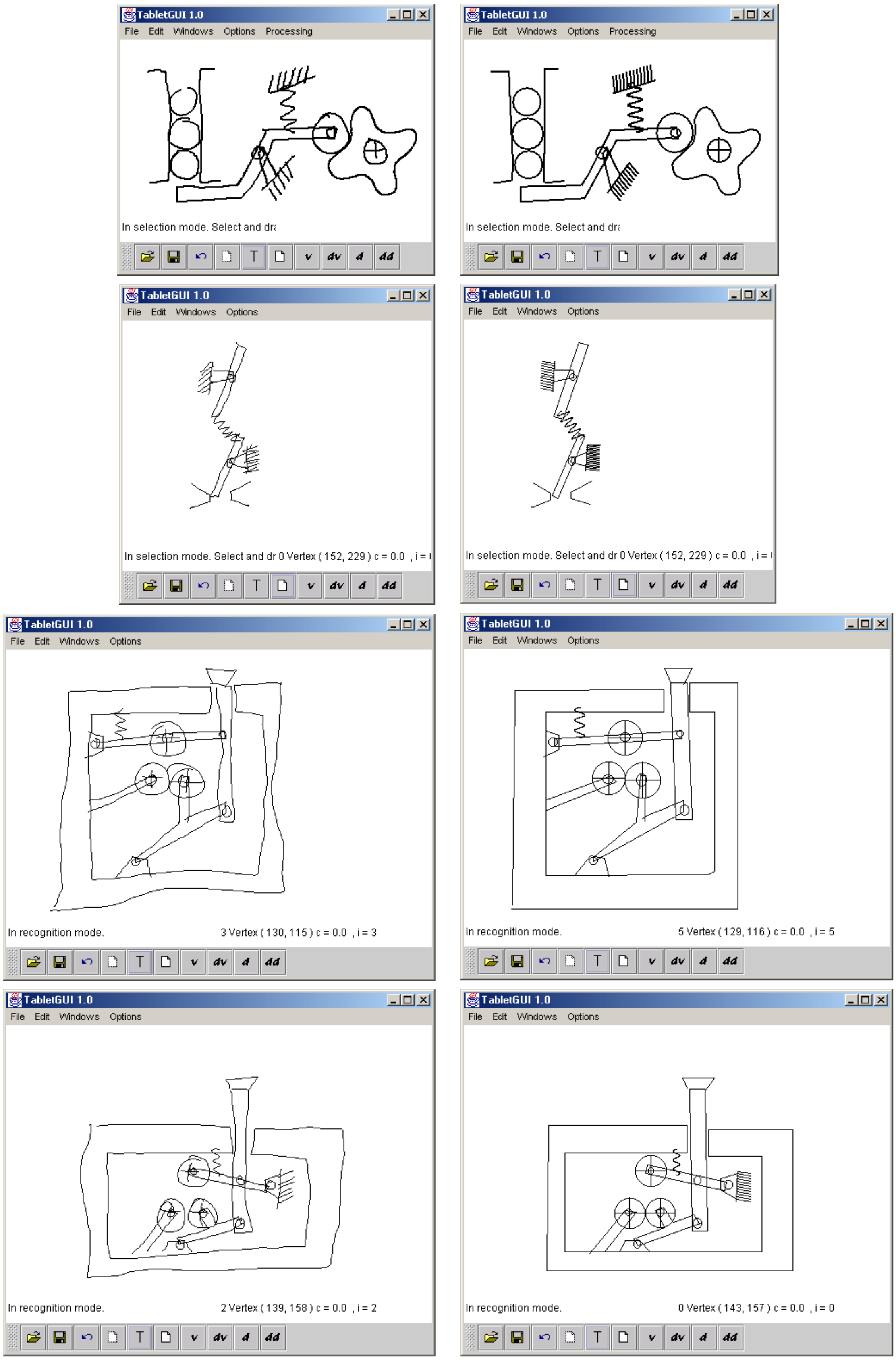


Figure 6-3: Performance examples: The first two pair are sketches of a marble dispenser mechanism and a toggle switch. The last two are sketches of the direction reversing mechanism in a tape player.

Chapter 7

Related Work

In this chapter, we will compare our system with some of the systems that support free-hand sketching. In general, these systems lack either one or more of the following properties that we believe a sketching system should have:

- It should be possible to draw arbitrary shapes with a single stroke, (i.e., without requiring the user to draw objects in pieces).
- The system should do automatic feature point detection. The user should not have to specify vertex positions by hand
- The system should not have sketching modes for drawing different geometric object classes (i.e., modes for drawing circles, polylines, curves etc.).
- The sketching system should feel natural to the user.

The Phoenix sketching system [18] had some of the same motivation as our work, but a more limited focus on interactive curve specification. While the system provided some support for vertex detection, its focus on curves led it to use Gaussian filters to smooth the data. While effective for curves, Gaussians tend to treat vertices as noise to be reduced, obscuring vertex location. As a result the user was often required to specify the vertices manually in [18].

Work in [6] describes a system for sketching with constraints that supports geometric recognition for simple strokes (as well as a constraint maintenance system and extrusion for generating solid geometries). The set of primitives is more limited than ours: each stroke is interpreted as a line, arc or as a Bézier curve. More complex shapes (e.g., squares, polylines) can be formed by combinations of these primitives, but only by user lifting the pen at the end of each primitive stroke, reducing the feeling of natural sketching.

The work in [4] describes a system for generating realtime spline curves from interactively sketched data. They focus on using knot removal techniques to approximate strokes known to be composed only of curves, and do not handle single strokes that contain both lines and curves. They do not support corner detection, instead requiring the user to specify corners and discontinuities by lifting the mouse button, or equivalently by lifting the pen. We believe our approach of automatically detecting the feature points provides a more natural and convenient sketching interface.

Zeleznik [8] describes a mode-based stroke approximation system that uses simple rules for detecting the drawing mode. The user has to draw objects in pieces, reducing the sense of natural sketching. Switching modes is done by pressing modifier buttons in the pen or in the keyboard. In this system, a click of the mouse followed by immediate dragging signals that the user is drawing a line. A click followed by a pause and then dragging of the mouse tells the system to enter the freehand curve mode. This approach of using modifier keys or buttons simplifies the recognition task significantly but puts extra burden on the user side. Our system allows drawing arbitrary shapes without any restriction on how the user draws them. There is enough information provided by the freehand drawing to differentiate geometric shapes such as curves, polylines, circles and lines from one another, so we believe requiring the user to draw things in a particular fashion is unnecessary and reduces the natural feeling of sketching. Our goal is to make computers understand what the user is doing rather than requiring the user to sketch in a way that the computer can understand.

Among the large body of work on beautification, Igarashi et al. [9] describes a system combining beautification with constraint satisfaction, focusing on exploiting features such as parallelism, perpendicularity, congruence and symmetry. The system infers geometric constraints by comparing the input stroke with previous ones.

Because sketches are inherently ambiguous, their system generates multiple interpretations corresponding to different ways of beautifying the input, and the most plausible interpretation is chosen among these interpretations. The system is interactive, requiring the user to do the selection, and doesn't support curves. It is, nevertheless, more effective then our system at beautification. However, beautification is not the main focus of our work and is present for the purposes of completeness.

Among the systems described above, the works in [18] and [4] describe methods for generating very accurate approximations to strokes that are known to be curves. The precision of these methods are several orders of magnitude below the pixel resolution. The Bézier approximations we generate are less precise but they are sufficiently precise for approximating free-hand curves. We believe techniques in [18] and [4] are excessively precise for free-hand curves, and the real challenge is detecting curved regions in a stroke as opposed to approximating those regions down to the numeric precision of the machine on which the system runs.

In the scale space community, the work in [14] describes a scale space based approach to dominant point detection. They also analyze corner interactions during the smoothing process. Our approach differs from their work in several aspects. In this work, we utilize curvature as well as speed data for feature point detection and we use scale space techniques in both settings. Furthermore, as we pointed out before, we derive the curvature scale space with single pass convolutions at each scale. The method used for deriving the scale space in [14] is twice as expensive compared to ours because they convolve x and y positions of the points separately and then derive the curvature.

The work presented here overlaps to an extent with the extensive body of work on document image analysis generally (e.g., [3]) and graphics recognition in particular (e.g., [19]), where the task is to go from a scanned image of, say, an engineering drawing, to a symbolic description of that drawing.

Differences arise because sketching is a realtime, interactive process, and we want to deal with freehand sketches, not the precise diagrams found in engineering drawings. As a result we are not analyzing careful, finished drawings, but are instead attempting to respond in real time to noisy, incomplete sketches. The noise is different as well: noise in a freehand sketch is typically not the small-magnitude randomly distributed variation common in scanned documents. In addition, information about pen's motion is a very useful information source available in online sketching systems, but not in scanned drawings.

Chapter 8

Future Work

We see three main directions for future work: making improvements to the current system, carrying out user studies, and integrating this system with other systems that require stroke approximation functionality.

8.1 Potential improvements

One of the weaknesses of our curve detection algorithm is that it relies on the feature point detection stage having a low false positive rate on the curved regions, but the methods we have described do not ensure this. Although we found the performance of curve detection to be satisfactory empirically, it would be nice to prune out the false positives on curved regions. We believe it is possible to filter out these false positives by looking at how much the curvature/speed data vary within the region of support for each feature point at the scale chosen by the algorithm we presented.

As an example, we revisit the stroke from the feature point detection chapter reproduced in Fig. 8-1 for convenience. The absolute value of the curvature data at the scale selected by our algorithm is given in Fig. 8-2. In this graph, the region corresponding to the larger of the two curved regions in the original stroke is between indices 50 and 100. As seen in this graph, visually detecting the local maxima in the this region is hard, because the variation in the function is too small. Note how little the function actually varies over the region of support for the false maxima compared

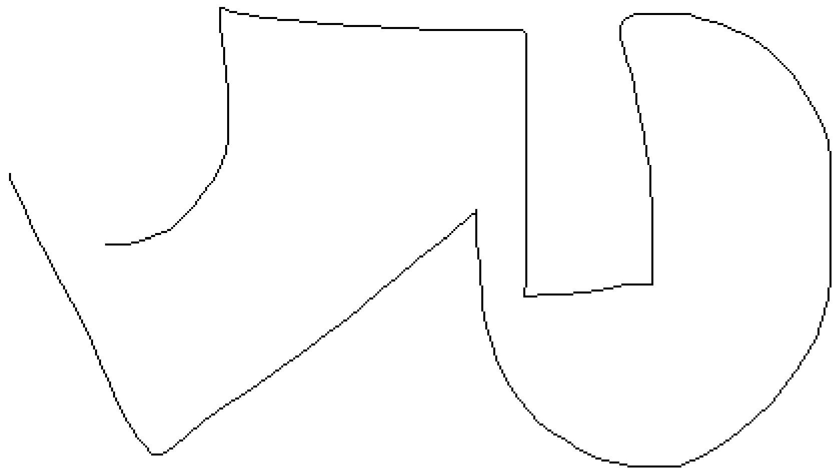


Figure 8-1: The stroke consisting of curved and straight segments revisited.

to the true ones. This property of the false maxima is also mentioned in [14]. We believe this property of the false positives can be used to eliminate false maxima by the choice of an appropriate threshold.

In the scale space literature, some authors proposed scale selection methods for computer vision tasks. In particular, in [11] and [12] Lindeberg describes how what he calls *the normalized* γ *derivatives* can be used to guide the scale selection in edge and ridge detection. We plan to explore whether this technique can be adapted for the problem of feature point detection and curve approximation.

8.2 User studies

User studies require choosing a number of domains where users sketch extensively and asking users to sketch naturally as they would with pencil and paper. The studies would measure the degree to which the system is natural i.e., supplies the feeling of freehand sketching while still successfully interpreting the strokes.

Another interesting task would be to observe how designers' sketching styles vary during a sketching session and how this may be used to improve recognition by perhaps introducing sketching modes. For example, humans naturally seem to slow down when they draw things carefully as opposed to casually. It would be interesting to conduct

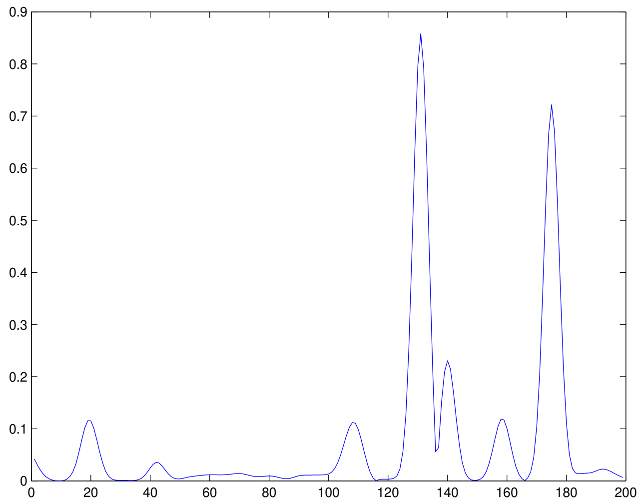


Figure 8-2: The curvature data for the curved stroke example at the scale chosen by our algorithm indexed by the point index on the x axis, and the curvature values in the y axis. Curvature between the indices 50 and 100 corresponds to the longer of the two curved regions in the stroke.

user studies to verify this observation and explore the degree to which one can use the time it takes to draw a stroke as an indication of how careful and precise the user meant to be. Then, it may be possible to define sketching modes and switch between these modes depending on user's behavior. Combining this idea with machine learning methods may lead to interesting results and improvements.

8.3 Integration with other systems

We are also trying to integrate this system to other work in our group that has focused on higher level recognition of mechanical objects [1]. This will provide the opportunity to add model-based processing of the stroke, in which early operations like vertex localization may be usefully guided by knowledge of the current best recognition hypothesis.

Yet another future direction would be to combine this work with some of the learning research to enable classifying a stroke using learned patterns rather than the template matching approach currently employed. We believe that our system may simplify stroke classification considerably by providing the learning engine with concise representation of input strokes.

Bibliography

[1] Christine Alvarado. A natural sketching environment: Bringing the computer into early stages of mechanical design. Master's thesis, Massachusetts Institute of Technology, 2000.

[2] J. Babaud, A. P. Witkin, M. Baudin, and R. O. Duda. Uniqueness of the gaussian kernel for scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:26–33, 1986.

[3] H. S. Baird, H. Bunke, and K. Yamamoto. Structured document image analysis. Springer-Verlag, Heidelberg, 1992.

[4] M. Banks and E. Cohen. Realtime spline curves from interactively sketched data. In *SIGGRAPH, Symposium on 3D Graphics*, pages 99–107, 1990.

[5] A. Bentsson and J. Eklundh. Shape representation by multiscale contour approximation. *IEEE PAMI 13, p. 85–93, 1992.*, 1992.

[6] L. Eggli. Sketching with constraints. Master's thesis, University of Utah, 1994.

[7] P. Boggs et al, editor. *User's Reference Guide for ODRPACK Version 2.01 Software for weighted Orthogonal Distance*. 1992.

[8] R. Zeleznik et al. Sketch: An interface for sketching 3d scenes. In *Proceedings of SIGGRAPH'96*, pages 163–170, 1996.

[9] T. Igarashi et. al. Interactive beautification: A technique for rapid geometric design. In *UIST '97*, pages 105–114, 1997.

[10] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.*

[11] T. Lindeberg. Feature detection with automatic scale selection.

[12] T. Lindeberg. Edge detection and ridge detection with automatic scale selection. *ISRN KTH/NA/P–96/06–SE, 1996.*, 1996.

[13] Micheal Oltmans. Understanding naturally conveyed explanations of device behavior. Master's thesis, Massachusetts Institute of Technology, 2000.

[14] A. Rattarangsi and R. T. Chin. Scale-based detection of corners of planar curves. *IEEE Transactionsos Pattern Analysis and Machine Intelligence*, 14(4):430–339, April 1992.

[15] N. Redding. Implicit polynomials, orthogonal distance regression, and closest point on a curve. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 191–199, 2000.

[16] R. Rosin. Techniques for assessing polygonal approximations of curves. *7th British Machine Vision Conf., Edinburgh*, 1996.

[17] Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25(4):329–337, 1991.

[18] P. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988.

[19] K. Tombre. Analysis of engineering drawings. In *GREC 2nd international workshop*, pages 257–264, 1997.

[20] David G. Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.

[21] A. Witkin. Scale space filtering. *Proc. Int. Joint Conf. Artificial Intell., held at Karsruhe, West Germany, 1983, published by Morgan-Kaufmann, Palo Alto, California*, 1983.

[22] A. L. Yuille and T. A. Poggio. Scaling theorems for zero crossings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:15–25, 1986.