11-25-2013 12:00 AM

Representation, Recognition and Collaboration with Digital Ink

Rui Hu
The University of Western Ontario

Supervisor
Dr. Stephen M. Watt
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Representation, Recognition and Collaboration with Digital Ink

(Thesis format: Monograph)

by

Rui Hu

Graduate Program
in
Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Pen input for computing devices is now widespread, providing a promising interaction mechanism for many purposes. Nevertheless, the diverse nature of digital ink and varied application domains still present many challenges. First, the sampling rate and resolution of pen-based devices keep improving, making input data more costly to process and store. At the same time, existing applications typically record digital ink either in proprietary formats, which are restricted to single platforms and consequently lack portability, or simply as images, which lose important information. Moreover, in certain domains such as mathematics, current systems are now achieving good recognition rates on individual symbols, in general recognition of complete expressions remains a problem due to the absence of an effective method that can reliably identify the spatial relationships among symbols. Last, but not least, existing digital ink collaboration tools are platform-dependent and typically allow only one input method to be used at a time. Together with the absence of recognition, this has placed significant limitations on what can be done.

In this thesis, we investigate these issues and make contributions to each. We first present an algorithm that can accurately approximate a digital ink curve by selecting a certain subset of points from the original trace. This allows a compact representation of digital ink for efficient processing and storage. We then describe an algorithm that can automatically identify certain important features in handwritten symbols. Identifying the features can help us solve a number of problems such as improving two-dimensional mathematical recognition. Last, we present a framework for multi-user online collaboration in a pen-based and graphical environment. This framework is portable across multiple platforms and allows multimodal interactions in collaborative sessions. To demonstrate our ideas, we present InkChat, a whiteboard application, which can be used to conduct collaborative sessions on a shared canvas. It allows participants to use voice and digital ink independently and simultaneously, which has been found useful in remote collaboration.

Keywords: Digital ink, InkML, handwriting recognition, mathematical handwriting recognition, multimodal collaboration

Acknowledgements

First and foremost I would like to express my deep gratitude and respect to my supervisor, Dr. Stephen M. Watt. His vast knowledge, constructive guidance, unyielding support, and empowering inspiration have added considerably to my graduate experience. Without any of these, this thesis would not have been possible. I was also greatly impressed by his personality, particularly his enthusiasm, attitude and great sense of humour. These are the things that are hard to find in books but will be influential to my future career.

I thank my colleagues and friends at the Ontario Research Centre for Computer Algebra for their generous help and valuable advice. It was an enjoyable experience working with them. I am also grateful to the professors, Dr. John Barron, Dr. Mahmoud El-Sakka, Dr. Marc Moreno Maza, Dr. Roberbo Solis-Oba, and Dr. Olga Veksler, who have taught me throughout the years. Your lessons will be well-cherished.

Last, I wish to thank my family for their constant love and encouragement. For my parents who raised me with love and supported me in all my pursuits. For my wife who provided me considerate care during the final stage of the Ph.D. Thank you!

Contents

List of Figures

List of Listings

Chapter 1

Introduction

1.1 Overview

Digital ink technology has experienced a tremendous growth over the past years. It has now been widely used by a variety of devices including Tablet PCs, PDAs, touch sensitive whiteboard systems, cameras and even smart phones. These devices accept digital ink input and pass it on to recognition software applications for interpretation. Systems can organize digital ink into documents or messages that can be stored for later reference or exchanged with other collaboration participants. These processes involve representation, recognition, and collaboration with digital ink. In this thesis, we investigate these aspects of digital ink and make contributions to each of them. Accordingly, this thesis is divided into three parts pertaining to aspects of digital ink representation, recognition and collaboration, each of which plays a critical role in digital ink technology.

A good representation is a key to success. Digital ink today is supported by a variety of devices and is captured using a number of methods, including optical tracking, electromagnetism, radio frequency and other technologies. Meanwhile, in addition to pen position, more and more devices allow recording other important information, such as pen tip pressure and pen tilt angle, to support handwriting recognition and authentication. Together, this makes digital ink data hard to be structured and has severely limited the capture, transmission, processing and presentation of digital ink across heterogeneous devices. Although a number of formats have been proposed, they are typically proprietary, which are restricted to single platforms and are conse-

quently not widely accepted. To address this issue, it requires a compact format that can accurately and flexibly represent digital ink. Such format will benefit digital ink applications for efficient storage and processing. This will be discussed in Section 1.2.

Recognition is desirable by many digital ink applications as it can convert handwriting input into machine-understandable format and allow useful computations. One of the sub-problems in handwriting recognition is to interpret handwritten mathematics. This is a very challenging problem in that mathematical input is two-dimensional, with elements of both writing and drawing. Moreover, writing mathematical symbols typically does not follow simple baselines and the symbols may be written in different sizes, such as subscripts and superscripts. This makes it difficult to reliably identify the spatial relationships among symbols. To solve this problem, it requires an effective method that can accurately differentiate between fluctuations in positioning and intentional sub- or super-scripts, which in return will benefit handwriting recognition. This will be discussed in Section 1.3.

Pen input is conducive to writing in mathematics since most mathematical notations are two dimensional. According to a study [1], pen input of mathematics is about three times faster and two times less error-prone than standard keyboard and mouse input. We can improve the efficiency even further by incorporating it into a multi-party, collaborative environment where multiple participants, who may be geographically separated, could write or draw, for instance, on a shared canvas. This has great potential to enhance distance collaboration between mathematicians, allowing them to share a common virtual space and interact with it in the most natural ways, just like having a discussion in the same room. With the widespread availability of pen-enabled devices, developing such a collaborative environment has garnered much interest from researchers in academia. This will be discussed in Section 1.4.

Within the broad areas of representation, recognition and collaboration, this thesis investigates a number of specific sub-problems. We identify these briefly and then continue with a general discussion of the context in which they arise.

- Chapter 2 describes an algorithm to obtain compact representation of digital ink.
- Chapter 3 investigates the suitability of InkML to flexibly represent sophisticated digital ink and proposes two brush models to demonstrate our ideas.
- Chapter 4 explores how to exchange digital ink data between different formats

and provides a data binding solution for two-way conversion between SketchML [2] and InkML [3].

- Chapter 5 proposes an algorithm to identify certain important metrics of handwritten mathematical symbols.
- Chapter 6 evaluates the performance of using homotopy methods to identify features in handwritten mathematical symbols.
- Chapter 7 investigates digital ink portability, both of digital ink data and of digital ink application code.
- Chapter 8 examines the design issues in incorporating multimodal interactions in mathematical collaboration.
- Chapter 9 proposes a streaming digital framework that is suitable for multi-party collaboration.
- Chapter 10 presents InkChat, a whiteboard application, to demonstrate our ideas.

It is expected that, by solving these sub-problems, it will greatly benefit digital ink applications. In the remainder of the chapter, we will further introduce each of the sub-problems, respectively.

1.2 Representation of Digital Ink

Data Representation

Existing handwriting recognition methods are typically based on representing characters by sequences of points, each of which contains x and y values in a rectangular coordinate system. This representation is readily available from a digital pen which samples points from a traced curve with a certain frequency and outputs the sequence of their coordinates in real time.

In order to accommodate detailed analysis and high definition rendering, higher sampling rates are used, allowing more points to be collected within an interval of time. However, this makes the input data lengthy and more costly to process. To solve

this problem, we present an algorithm that can accurately approximate a digital ink curve through selecting a subset of points from the original trace. In particular, the algorithm can find an approximation with a specified number of points, which yields the minimum cumulative error. Alternatively, it may be used to select the minimum number of points required to satisfy an error bound. This algorithm uses dynamic programming. The main advantage of the algorithm is that it is independent of the choice of compatible error function and has a cost linear in the number of points. This algorithm can be used to compact the representation of digital ink data while preserving the shape of curves.

A flexible format that can accurately represent digital ink is essential for pen-enabled software applications. Modern devices have reached maturity to support different drawing and writing activities, e.g. diagramming and digital painting, by providing additional information such as pen tilt angle, pen tip force, timestamp, *etc.* However, few digital ink formats allow recording the additional information. After considering various alternatives, we have arrived at a design of using InkML [3] to represent digital ink. InkML has the advantage of being a W3C standard format. It provides application-defined channels to support sophisticated digital ink representation. To demonstrate our ideas, we present two virtual brush models, Round Brush and Tear Drop Brush. Both models can be used to improve the rendering quality of digital ink. The Tear Drop Brush has been adopted into the InkML standard at our suggestion.

We also note that with the widespread availability of pen-enabled devices it is becoming increasingly important to be able to exchange drawing and writing input between platforms and applications. However, various vendors record pen or touch input either in their own proprietary formats, or simply as images that lose important information. A number of formats for digital ink have been proposed earlier, including Unipen [4, 5] and JOT [6], but these have either been restricted to special uses or have not received wide acceptance. Presently two data formats stand out as sufficiently general for wide-spread use: *MIT SketchML* and *InkML*. Techniques for exchanging digital ink data between the two formats, however, are currently not readily available. To address this issue, we describe a data binding solution for two-way conversion between SketchML and InkML. We show how to transform SketchML files to InkML archiving and streaming style. This allows digital ink data to be used in collaborative environments where real-time sharing is desired. In the reverse conver-

sion, we bind InkML elements to SketchML elements. This makes digital ink data that is represented by InkML available to existing SketchML applications.

Mathematical Representation

As the sampling rate and resolution vary between different vendors and over generations of technology, such data is, however, not device-independent and therefore depends on the specific capturing hardware. In order to use data from different capture devices, various treatments have been developed, including "re-sampling" (interpolation), which try to convert device resolution after the fact. These methods replace the captured data with made up data that might have been captured on a different device. To address this issue, we adopt an approach that can represent handwritten symbols in the space of coefficients of a functional approximation. This approach has been used in earlier work [7, 8, 9, 10].

We consider an ink trace as a segment of a plane curve $(x(s), y(s))$, parameterized by Euclidean arc length, s,

$$ds^2 = dx^2 + dy^2$$

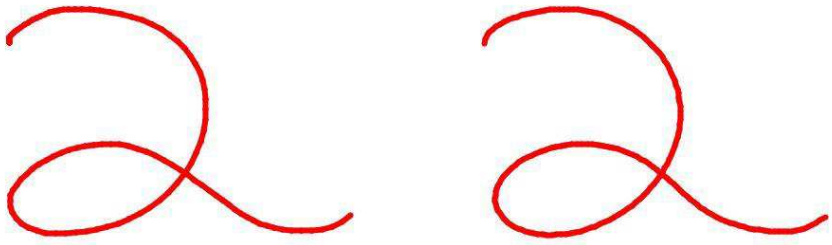
This parameterization has been found to lead to good recognition and makes intuitive sense, since it gives curves that look the same the same parameterization [7]. Given a digital ink trace $(x(s), y(s))$ and an approximating basis $\{B_i(s)\}_{i=0,\ldots,d}$, we represent the trace using the coefficients x_i and y_i from

$$x(s) \approx \sum_{i=0}^{d} x_i B_i(s) \qquad y(s) \approx \sum_{i=0}^{d} y_i B_i(s)$$

It is convenient to choose the functions $B_i(s)$ to be orthogonal polynomials, e.g. Chebyshev, Legendre or some other polynomials. By choosing an appropriate family of basis polynomials to high enough degree, the approximating curve can be made arbitrarily close to the original trace.

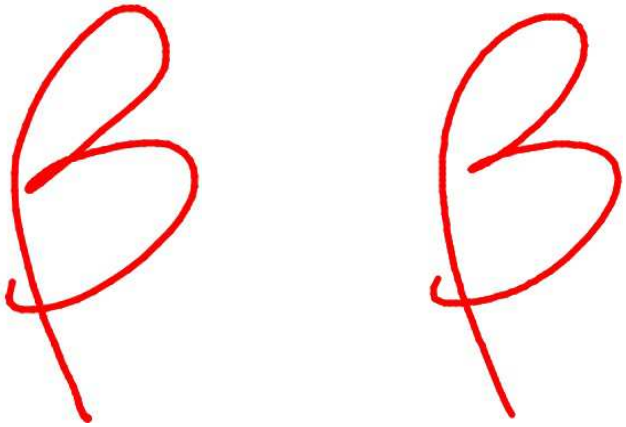
We have found a Legendre-Sobolev basis allows approximating curves to have the desired shape for relatively low degrees. These may be computed by Gram-Schmidt orthogonalization of the monomials $\{s^i\}$ with respect to the inner product

$$\langle f, g \rangle = \int_a^b f(s)g(s)\mathrm{d}s + \mu \int_a^b f'(s)g'(s)\mathrm{d}s.$$



(a) (b)

Figure 1.1: Approximation using Legendre-Sobolev series. (a) Original. (b) Approximated using series of degree 12 with $\mu = 1/8$.



(a) (b)

Figure 1.2: Another example of approximation using Legendre-Sobolev series. (a) Original. (b) Approximated using series of degree 12 with $\mu = 1/8$.

If a symbol has multiple strokes, we join consecutive strokes by concatenating the point series, which yields a single curve. For more details see [10]. Examples of using Legendre-Sobolev polynomials in approximation is shown in Figure 1.1 and Figure 1.2. After approximation, we may now represent the digital ink trace, or symbol, as the coefficient vector $(x_0, ..., x_d, y_0, ..., y_d)$. We may standardize the location and size of the character by setting x_0, y_0 to 0 and the norm of the vector to 1. Such representation is more robust under changes in hardware and can consequently make handwriting recognition software more portable.

1.3 Recognition of Digital Ink

Recognition is essential for digital ink applications as it can interpret the input and allow useful computations. Current systems are now achieving good recognition rates on individual mathematical symbols, in general recognition of complete expressions remains a problem due to its two-dimensional nature. In handwritten mathematics, it is common to have symbols in various sizes and for the writing not to follow simple baselines. For example, subscripts and superscripts appear relatively smaller than normal text and are written slightly below or above it. Moreover, these subscripts and superscripts may themselves have subscripts and superscripts. Such notation makes the analysis of the spatial relationships between symbols challenging as it introduces various ambiguities. For example, whether a particular symbol is lowercase "p" or an uppercase "P" makes the difference between a subscripted p_q or the juxtaposed Pq. To help solve the problem, it requires an effective algorithm that can accurately differentiate between fluctuations in positioning and intentional sub- or super-scripting over a baseline. In order to accomplish this, we need to understand the scale and relative positioning of individual symbols. Therefore, it becomes necessary to identify the location of certain expected features, such as the location of baseline, which are typically defined by particular points in the symbols. These particular points occur at different locations in different symbols, and their precise locations may vary in different handwriting samples of the same symbol. For example, the baseline of a lower "p" would be identified by the lowest part of the bowl, ignoring the descender. In contrast, the baseline of lowercase "k" would be identified by the toes. We refer to a point such as this, that determines the height of a metric line, as a *determining point*.

Finding determining points for an individual handwritten symbol is easy. One can simply annotate the symbol with the positions of all those points. However, identifying determining points for all symbols in a collection or symbols recovered as input to an application is much more challenging. First, it is prohibitively costly to manually annotate a large database. Secondly, applications such as mathematics involve a large variety of symbols derived from a range of alphabets and other sources. In practice, many of them are often poorly written and there is no fixed dictionary of words to assist in disambiguation [11]. This increases the difficulty to find determining points reliably. Meanwhile, each person's handwriting is unique — even identical twins write differently [12]. Even if a training database were to be fully annotated, it

is not entirely clear how this should best be used to identify the points of interest in new input. Last, but not least, the usual methods for detecting determining points depend on device resolution significantly. With rapidly evolving technology, this means that new algorithms cannot use archival data directly and therefore must be "re-sampled" (interpolated).

To address this problem, we present an algorithm that can automatically identify the determining points in handwritten symbols. Knowing the determining points of each symbol can help us solve a number of problems. For example, one can use the determining points to improve two-dimensional mathematical recognition. By comparing the baseline locations and the sizes of adjacent symbols, one can identify subscripts and superscripts (e.g. S_2, $S2$, S^2) with more confidence. Another application is in handwriting neatening. Since handwritten symbols often come with variations in alignment and size, certain transformations based on determining points can be applied to obtain normalized samples while preserving the original writing style. In contrast to existing methods, which treat digital ink traces as a collection of discrete points, this algorithm relies on interpreting ink traces as single points in a functional space. This allows device independence, on one hand, and a simple formulation of homotopic deformation, on the other. To evaluate the performance of the algorithm, we test it against a database of handwritten mathematical characters. The experiments show promising results.

We learned from the earlier experiments that for characters that are significantly different from the average instance, one can use a numerical homotopy between the model instance and the target character, and apply a local extremum-finding algorithm at each step. We further investigate the factors to be taken into account in performing such homotopies. We examine two strategies for possible starting points for the homotopy, and the relation between the distance from the starting points to the ending points and the number of steps required. The first strategy performs a homotopy to the average symbol, constructed from all samples of the same type. The second strategy uses a homotopy to the nearest neighbor with known determining points. We present our experimental results against a database of handwritten mathematical characters. The results suggest improved strategies to find determining points for poorly written characters.

1.4 Collaboration with Digital Ink

We are motivated by the problem of how to support computer-enhanced distance collaboration, where machines can interpret the input and allow useful computations. However, this problem presents many challenges. First, existing techniques typically use Application Program Interfaces (APIs) and proprietary ink formats that are restricted to single platforms and consequently lack portability. Collaboration software, however, is most useful when it is platform-independent, both because an individual may use different platforms at different times, and because teams may be composed of members using different systems. Dealing with significantly different client software reduces the ability of the individual or the group to master its use. If any of the members has difficulty, efficiency of the whole team suffers. Second, existing systems typically allow only one input method to be used at a time. For example, in Microsoft OneNote, one can either type or draw, but not simultaneously. This places strong limitations on what can be done. For example, it becomes quite awkward to explain an activity while it is being performed. Last, but not least, we believe that the synergy of pen-based collaboration and recognition of mathematical input can enhance the efficiency of online interaction. Nevertheless, there is no technology that allows to capitalize on both simultaneously: some software handles recognition without the ability for real-time sharing, e.g. the Maple computer algebra system [13], while others provide sharing, but no mathematical recognition, e.g. Microsoft OneNote [14], Calliflower [15] or Dabbleboard [16].

To address these issues, we first present a portable framework that can handle the details of a variety of platforms and provide a consistent, platform-independent interface for digital ink applications. Applications that are built on top of the framework can collect digital ink across all the supported platforms without knowing their underlying details. In addition, this framework uses InkML as data representation as it provides both digital ink streaming and archival support independent of platforms. Together, it makes portable digital ink applications possible so that we can "write once and run anywhere". We then investigate the question of how multimodality can be used in computer-mediated mathematical collaboration. We analyze the design issues in incorporating multimodal interactions in this kind of collaboration. Finally, we propose a framework for multi-user online collaboration in a pen-based and graphical environment. This environment allows participants conducting and archiving collaborative sessions that involve synchronized voice and digital ink on a

shared canvas. The digital ink is represented as InkML, allowing special recognizers for different content types, such as mathematics and diagrams. The collaborative sessions may be recorded and stored for later playback, analysis or annotation. To demonstrate our ideas, we present InkChat, a whiteboard application, which can be used to conduct collaborative sessions on a shared canvas. It allows participants to use voice and digital ink independently and simultaneously, which has been found useful in mathematical collaboration.

Part I

Representation

Chapter 2

Compact Representation of Digital Ink

Digital ink curves are typically represented as series of points sampled at certain time intervals. We are interested in the problem of how to select a minimal subset of sample points to approximate a digital ink curve within a given error bound. We present an algorithm to find an approximation with a specified number of points and providing the minimum cumulative error. Alternatively, it may be used to select the minimum number of points required to satisfy an error bound. This chapter is based on the article "Optimization of Point Selection on Digital Ink Curves" [17] co-authored with Stephen M. Watt, that appeared in the proceedings of the 13th International Conference on Frontiers in Handwriting Recognition.

2.1 Introduction

Pen input is one of the more convenient and natural forms of entry for several kinds of input, and has therefore been adopted by a variety of electronic devices such as tablets, PDAs, touch sensitive whiteboards, and cell phones. Tied to the pen input is the notion of digital ink. Digital ink is generated by sampling points from a traced curve at a certain rate, and thus is typically presented in the form of a series of points, each of which contains x and y values in a rectangular coordinate system at a particular time. Recognition software applications take these sequences as input. In order to accommodate detailed analysis and high definition rendering,

higher sampling rates are used, allowing more points to be collected within an interval of time. However, this creates more work for recognition software applications and demands more resources for storage. We are therefore interested in how to select a subset of these sampled points that retain a desired degree of accuracy.

We are motivated by the problem of how to accurately approximate a digital ink curve through selecting a subset of points from the original trace. We wish to reduce the size of the subset while bounding the approximation's error. In other words, we would like to save the critical points that determine the shape of the curve (e.g. turning points) and remove those which have little impact (e.g. middle points on a straight line).

2.2 Previous Work

This is a problem for which there has been considerable previous work, some of which we highlight here. In 1986, Dunham [18] proposed an optimal algorithm to find a piecewise linear approximation with fixed initial and final points by selecting a subset of points from the original set. The approximation is optimal in the sense that it contains the minimum number of segments such that the error on each is below a uniform threshold. The error on each segment was taken to be the maximal distance from the curve segment to the line segment. In 1996, Horst and Beichl [19] introduced an algorithm which used arc-chord length difference as the error. Compared to [18], this algorithm achieved lower complexity but cannot guarantee global optimality. In 2007, another algorithm was presented in [20], which iteratively computes chordal deviation—the distance between the original curve and its approximation. Points with the minimal distance are removed until the distance becomes larger than a threshold. In 2012, Mazalov and Watt [21] described a piecewise linear approximation algorithm to compress digital ink. That algorithm is fast but suboptimal and selects points using a combination of two error functions. All of these algorithms compute the error on each curve segment and attempt to minimize the maximum error. They do not minimize the cumulative error which reflects the approximation's global deviation from the original curve. Sometimes the maximum error on each curve segment can be small but the cumulative error large, which can produce global distortion.

2.3 Objectives

Our method is based on the observation that, for piecewise linear approximation, removing sample points gives approximating curves of shorter arc length. Arc length discrepancy is additive and may be used as a proxy for other error measures. A continuous curve may be approximated by a piecewise linear function with vertices on the curve. Decreasing the arc length discrepancy by adding points to a piecewise linear approximation decreases all the usual error measures and cannot increase them.

We present an algorithm to find an optimal point selection to approximate a piecewise linear curve. It can be used in two ways:

- Given a digital ink curve consisting of $n \geq 2$ points and a specified number of points $2 \leq k \leq n$, the algorithm selects a subset of k points such that the arc length discrepancy between the approximation and the original curve is minimized.
- Given a digital ink curve and a bound on arc length discrepancy, the algorithm selects a subset of points of minimum number required to approximate the curve to within that bound. That is, no smaller subset of the original points can achieve the bound.

Both uses are globally optimal and can be applied to both open and closed, planar and space curves.

The method can be applied when other error measures are of interest. In this case, though fast and good, the point selection is not guaranteed to be optimal. We have used this method with a variety of error types used in prior work, including the arc-chord length difference, maximal height, average height, which in turn measure the difference between the curve length and the chord length, the maximal height from the curve to the chord, and the average height from the curve to the chord. All of these errors are computed on each curve segment.

The remainder of the chapter is organized as follows. In section 2.4, we present the algorithm and analyze its correctness and complexity. Section 2.5 reports on experiments conducted to evaluate the performance of with several error functions. Section 2.6 summarizes the chapter.

2.4 The Approximation Algorithms

2.4.1 Problem Definition

We consider a digital ink curve to be a two dimensional curve made up of a series of points. Our objective is to find an acceptable approximation by selecting a subset of points from the original ones. We have two problems:

- Given a digital ink curve consisting of $n \geq 2$ points and a specified number of points k, $2 \leq k \leq n$, how can we select the k points such that the cumulative error between the approximation and the original curve is minimized?
- Given a digital ink curve consisting of $n \geq 2$ points and a cumulative error threshold $\epsilon \geq 0$, how can we select the minimum number of points required to approximate the curve such that the cumulative error is less than or equal to ϵ?

Both problems can be seen as graph problems. Given a digital ink curve consisting of n points, we first assign an index to each point. A weighted DAG (directed acyclic graph) $G(V, E)$ can be constructed from these points, where

$$\begin{cases} V &= \{v_i \mid 0 \leq i \leq n-1\} \\ E &= \{(v_i, v_j) \mid 0 \leq i < j \leq n-1\} \end{cases} \tag{2.1}$$

The set V contains n vertices, with v_i corresponding to the i-th point p_i on the digital ink curve. The DAG will be constructed to have a unique source (vertex with no inbound edge) and a unique sink (vertex with no outbound edge). The source corresponds to the initial point p_0 and the sink corresponds to the final point p_{n-1}. The weight of each edge is defined as:

$$w(v_i, v_j) = errorFn(p_i, p_j) \tag{2.2}$$

The error function $errorFn(p_i, p_j)$ is given beforehand. It measures the approximation error on the curve segment determined by p_i and p_j. Different error functions can be applied to compute for different error types. Section 2.4.3 will explain the error types in detail.

Algorithm 1: Approximation by k points

Input: A digital ink curve of n points, $n \geq 2$
Input: The specified number k, $2 \leq k \leq n$
Output: The indices of the k points
begin
  // The indices of the k points
  $S \leftarrow \{\}$;
  // The minimum weight table
  $D \leftarrow (k+1) \times n$ *matrix*;
  // Path
  $P \leftarrow (k+1) \times n$ *matrix*;
  // Initialization
  for $j \leftarrow 1$ **to** $n-1$ **do**
    $D_{2,j} \leftarrow w(v_0, v_j)$;
    $P_{2,j} \leftarrow 0$;
  // Compute the rest of D
  for $m \leftarrow 3$ **to** k **do**
    for $j \leftarrow m-1$ **to** $n-1$ **do**
      $min\_weight \leftarrow \infty$;
      for $i \leftarrow m-2$ **to** $j-1$ **do**
        $weight \leftarrow D_{m-1,i} + w(v_i, v_j)$;
        $prior\_vertex\_index \leftarrow 0$;
        if $weight < min\_weight$ **then**
          $min\_weight \leftarrow weight$;
          $prior\_vertex\_index \leftarrow i$;
      $D_{m,j} \leftarrow min\_weight$;
      $P_{m,j} \leftarrow prior\_vertex\_index$;
  // Restore the path
  $vertex\_index \leftarrow n-1$;
  for $i \leftarrow 0$ **to** $k-1$ **do**
    $S \leftarrow S \cup \{vertex\_index\}$;
    $vertex\_index \leftarrow P_{k-i,vertex\_index}$
  return S

The first problem is now equivalent to finding a path from the source to the sink consisting of k vertices with minimum total weight. The second problem is equivalent to finding the shortest path from the source to the sink such that the total weight is less or equal to the given threshold.

Algorithm 2: Approximation by error threshold

Input: A digital ink curve of n points, $n \geq 2$
Input: The error threshold ϵ, $\epsilon \geq 0$
Output: The indices of the selected points
begin
  // The selected points
  $S \leftarrow \{\}$;
  // The minimum weight table
  $D \leftarrow (n+1) \times n$ *matrix*;
  // Path
  $P \leftarrow (n+1) \times n$ *matrix*;
  // The smallest m that makes $D_{m,n-1} \leq \epsilon$
  $m^* = 2$;
  // Initialization
  for $j \leftarrow 1$ **to** $n-1$ **do**
    $D_{2,j} \leftarrow w(v_0, v_j)$;
    $P_{2,j} \leftarrow 0$;
  if $D_{2,n-1} > \epsilon$ **then**
    // Compute the rest of D
    for $m \leftarrow 3$ **to** n **do**
      for $j \leftarrow m-1$ **to** $n-1$ **do**
        $min\_weight \leftarrow \infty$;
        for $i \leftarrow m-2$ **to** $j-1$ **do**
          $weight \leftarrow D_{m-1,i} + w(v_i, v_j)$;
          $prior\_vertex\_index \leftarrow 0$;
          if $weight < min\_weight$ **then**
            $min\_weight \leftarrow weight$;
            $prior\_vertex\_index \leftarrow i$;
        $D_{m,j} \leftarrow min\_weight$;
        $P_{m,j} \leftarrow prior\_vertex\_index$;
      if $D_{m,n-1} \leq \epsilon$ **then**
        $m^* \leftarrow m$;
        break;
  // Restore the path
  $vertex\_index \leftarrow n-1$;
  for $i \leftarrow 0$ **to** m^*-1 **do**
    $S \leftarrow S \cup \{vertex\_index\}$;
    $vertex\_index \leftarrow P_{m^*-i,vertex\_index}$
  return S

2.4.2 Algorithm

Both paths are guaranteed to exist and can be found using dynamic programming. Given a graph $G(V, E)$, we define a matrix D, where $D_{m,j}$ represents the minimum total weight of the path from the source, v_0, to vertex v_j including m vertices. Initially, we assign

$$D_{2,j} = \begin{cases} \infty & \text{if } j = 0 \\ w(v_0, v_j) & \text{if } 0 < j \leq n-1 \end{cases}$$

For $m \geq 3$, $D_{m,j}$ can be computed as:

$$D_{m,j} = \begin{cases} \min\limits_{m-2 \leq i < j} \{D_{m-1,i} + w(v_i, v_j)\} & \text{if } j \geq m-1 \\ \infty & \text{otherwise} \end{cases}$$

Therefore, finding the minimum cumulative error of a k-point approximation is simply to compute $D_{k,n-1}$, where k is the specified number of points and $n-1$ is the index of the final point on the original curve. The complete algorithm to select the k points is shown in Algorithm 1.

Similar to Algorithm 1, finding an approximation consisting of the minimum number of points such that the cumulative error is within a given threshold, ϵ, is achieved by exiting the loop with a `break` statement. We keep computing $D_{m,n-1}$ for $m = 2 \ldots n$ until we find the first m that makes $D_{m,n-1} \leq \epsilon$. For additive errors, the m is guaranteed to exist as the cumulative error decreases when more points are selected and reaches 0 when all points are selected. The complete algorithm is laid out in Algorithm 2.

2.4.3 Error Types

We construct digital ink curves using linear and cubic spline interpolation methods since they are commonly used in the area of digital ink rendering, handwriting recognition, and handwriting neatening. By selecting a subset of points, the approximation algorithm introduces differences between the approximation and the original curve. These differences introduce error. The error is measured on each segment (i.e. the interval between each pair of points on the curve), and we assign zero to the error on the segment formed by any two consecutive points. Errors can be cumulated along

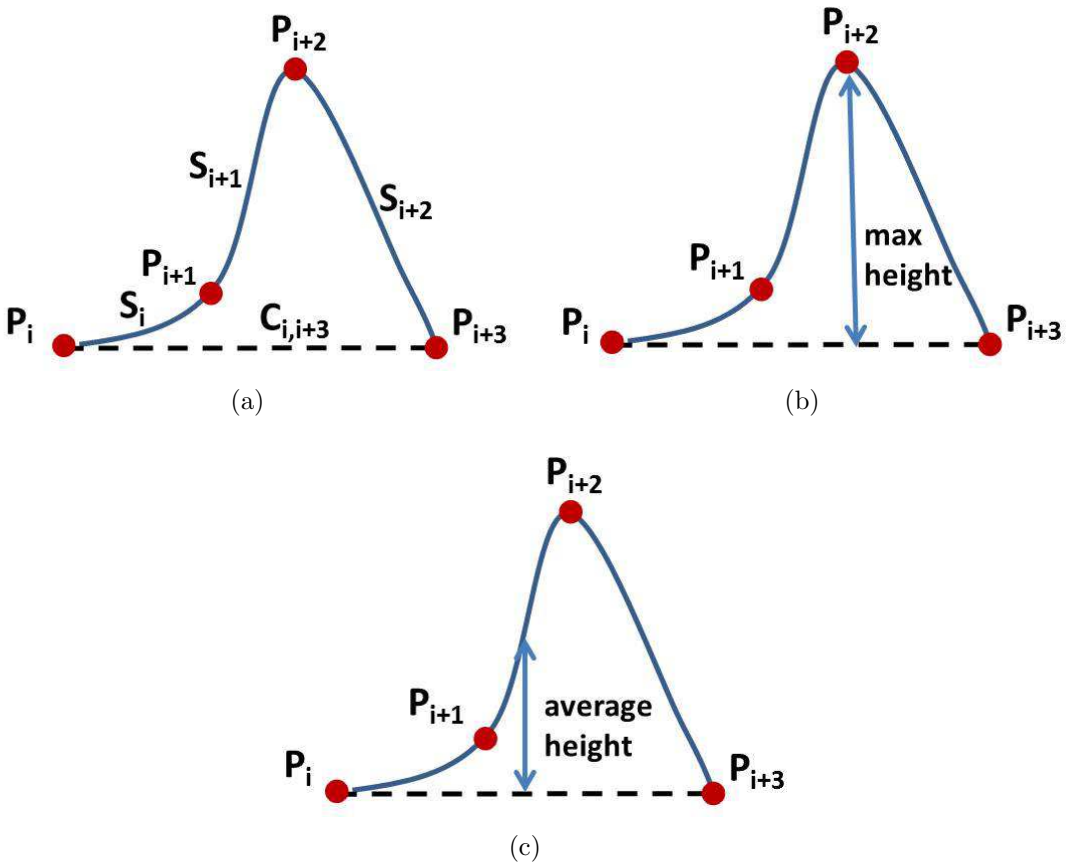


Figure 2.1: Error types: (a) Arc-Chord Length Error, (b) Maximal Height Error, and (c) Average Height Error. The curve is constructed using cubic spline interpolation.

the approximated curve, which reflects the global deviation from the original one. As digital ink curves may be generated in different scales, we normalize each by its arc length in order to evaluate the error fairly.

As we are interested in minimizing the global deviation error, we choose error types based on three criteria. A good type of error should be computationally efficient, additive, and has a natural meaning in geometry. In this thesis, we consider three types: Arc-Chord Length Error, Maximal Height Error, and Average Height Error.

- **Arc-Chord Length Error** measures the difference between the sum of the arc length of each curve piece and the length of the chord. An example is shown in Figure 2.1(a). The error on the segment (p_i, p_{i+3}) is computed as $S_i + S_{i+1} + S_{i+2} + S_{i+3} - C_{i,i+3}$, where S is the arc length of the curve piece and C is the chord length.

- **Maximal Height Error** measures the maximal distance between the curve segment and the line segment. An example is shown in Figure 2.1(b).
- **Average Height Error** measures the average distance between the curve segment and the line segment. An example is shown in Figure 2.1(c).

2.4.4 Correctness

Selecting the m-th point is a process of computing $D_{m,n-1}$, where $n-1$ is the index of the final point on the original digital ink curve. Since

$$D_{m,n-1} = \min_{m-2 \leq i < n-1} \{D_{m-1,i} + w(v_i, v_{n-1})\},$$

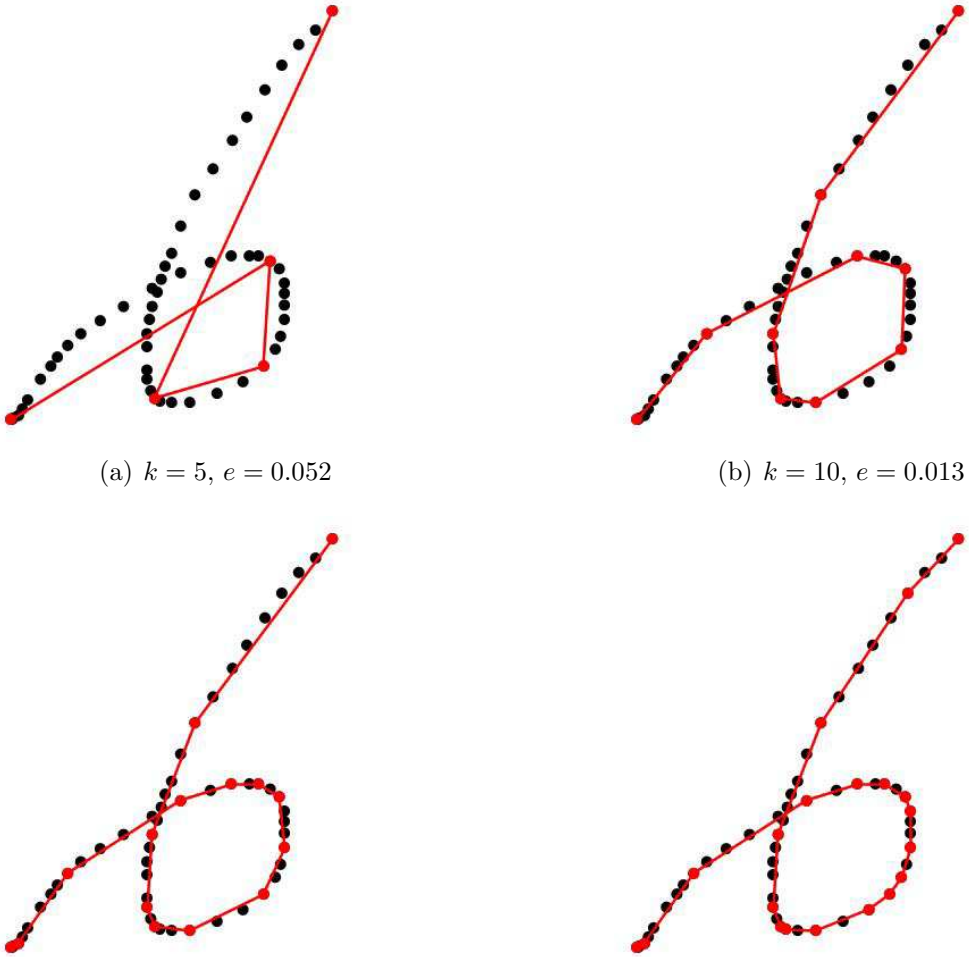
we can recursively compute $D_{m,n-1}$, $m = 3 \ldots n$ using dynamic programming with the given initial condition $D_{2,i} = w(v_0, v_i)$, $0 < i \leq n-1$. This is a particular application of the Principle of Optimality [22] and $D_{m,n-1}$ will be the minimum cumulative error of the approximation consisting of m points.

Since the Arc-Chord Length error is additive, the matrix D has the following properties: with the increase of m, $D_{m,n-1}$ decreases and reaches 0 when $m = n$. But for the other two types errors, the matrix D may not have the property that $D_{m,n-1} \leq D_{m',n-1}$ when $m' \geq m$. However, since $D_{n,n-1}$ is 0 in either case, we can always use the algorithm to find the solution.

2.4.5 Complexity

The complexity to find the k-point approximation is $O(kn^2)$. To see this, note we are computing a matrix. To select k points, it is necessary to compute k rows. Since each row has n entries, we have a total cost of $O(kn^2)$. Finding an approximation within a given cumulative error threshold ϵ, in the worst case that all points on the original digital curve need to be selected, giving complexity $O(n^3)$.

The both complexities can be reduced if we look at only a fixed number of prior vertices in computing $D_{m,j}$, but the approximation result may no longer be globally optimal.



(a) $k = 5$, $e = 0.052$

(b) $k = 10$, $e = 0.013$

(c) $k = 15$, $e = 0.005$

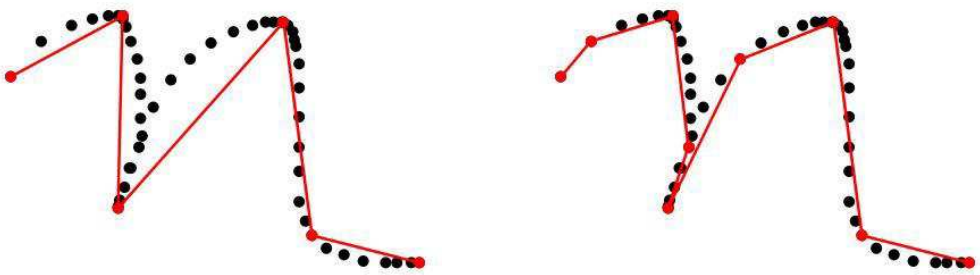
(d) $k = 20$, $e = 0.002$

Figure 2.2: Approximation of symbol "6" by specified number of points k. The black points are the points on the original curve. The red points are the approximation. The error is the arc-chord length error. We use e to denote the cumulative error.

2.5 Experiments

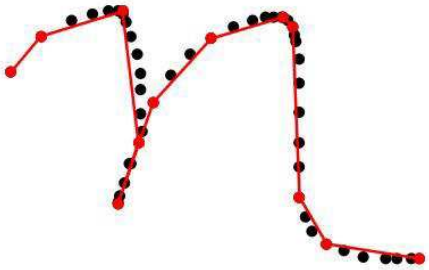
Figure 2.2 shows an example of applying Algorithm 1 to a digital ink curve. The digital ink curve consists of 55 points which are marked as black dots. The red dots are the selected points in the approximation. The error adopted here is the Arc-Chord Length Error. Increasing the number of selected points from 5 to 20, the approximation approaches the original digital ink curve quickly.

Figure 2.3 shows an example of applying Algorithm 2. The digital ink curve consists of 51 points which are marked as black dots. The red dots are the selected

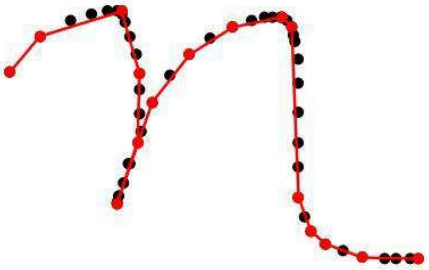


(a) $\epsilon = 0.05$, $s = 6$, $e = 0.0459$

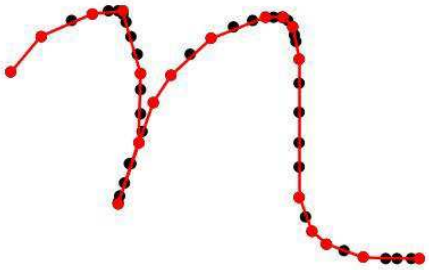
(b) $\epsilon = 0.02$, $s = 9$, $e = 0.0182$



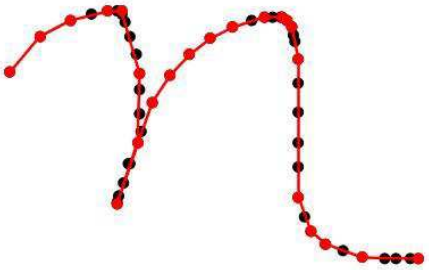
(c) $\epsilon = 0.01$, $s = 12$, $e = 0.0093$



(d) $\epsilon = 0.005$, $s = 16$, $e = 0.004$



(e) $\epsilon = 0.002$, $s = 19$, $e = 0.0018$



(f) $\epsilon = 0.001$, $s = 23$, $e = 0.0009$

Figure 2.3: Approximation of symbol "n" by cumulative error threshold ϵ. The black points are the points on the original curve. The red points are the approximation. The error is the Arc-Chord Length Error. We use s and e to denote the number of selected points and the cumulative error, respectively.

points in the approximation. The error adopted here is the Arc-Chord Length Error. As we decrease the error threshold ϵ, more points are selected in order to restrict the cumulative error within ϵ. When the error threshold drops to 0.001, the approximation is almost as the same as the original digital ink curve. But the number of points selected in the approximation is only half of the size of the original.

We have discussed three types of error in Section 2.4.3. To give a general idea of the performance for each error type, we have tested our algorithms against a handwriting dataset. The handwriting dataset we used is that of LaViola [23], containing 10665 symbols, mostly Latin letters and digits. Altogether there are 13203 strokes

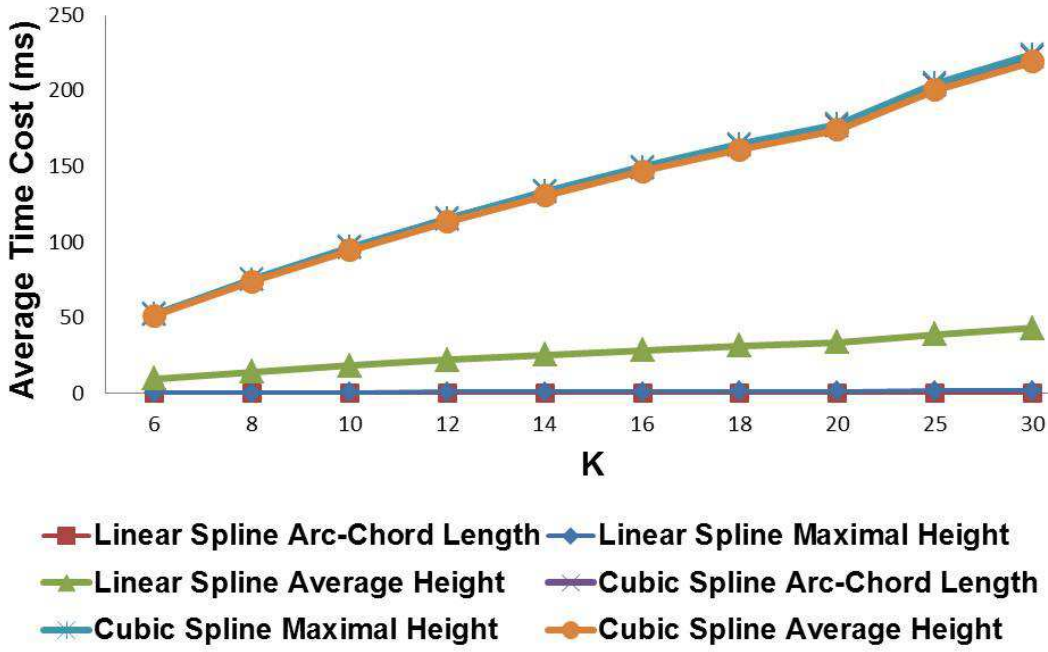


Figure 2.4: The average time cost of using different error types in Algorithm 1.

in these symbols. We constructed digital ink curves from these strokes using linear and cubic spline interpolation since they are commonly used in the area of digital ink rendering, handwriting recognition, and handwriting neatening. All of the curves were normalized by arc length in advance. We measured the average time cost in milliseconds and average cumulative error (relative to the trace length) on each digital ink curve. Figure 2.4 shows the average time cost of applying different error types in Algorithm 1. We see that the Arc-Chord Length Error outperforms the others in both linear and cubic spline cases. Figure 2.5 shows the average cumulative error of applying different error types in Algorithm 1. With the increase of the specified number of points, the average cumulative errors of all types drop dramatically, but takes longer to compute.

Evaluation of these error types can be conducted in different ways. Since the approximated curve and original curve share the same initial and final points, one can compare the area enclosed by the two curves. A smaller area indicates that the approximated curve in general looks more similar to the original one, which suggests a better approximation. When the enclosed area becomes 0, the approximated curve will overlap the original one and both curves will look exactly same. An alternative way to evaluate these error types is to compare the arc length between the approxi-

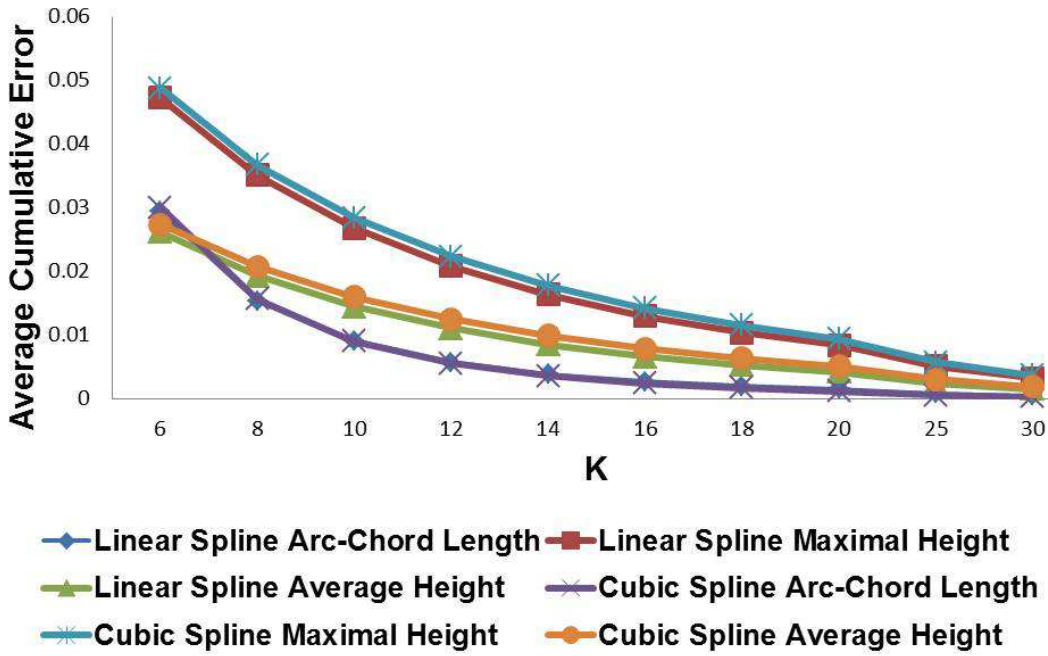


Figure 2.5: The average cumulative error of using different error types in Algorithm 1.

mated curve and original one. Since the two curves share the same initial and final points and our point selection is a subset of the entire point set, one can compute the difference between the arc length of the approximated curve and of the original curve. A smaller difference suggests a higher similarity and a better approximation. Figure 2.6 shows the average similarity which is defined as the arc length of the approximated curve divided by the arc length of the original curve. With the increase of the specified number of points, the arc length of the approximated curves approaches the arc length of original curve quickly.

2.6 Summary

We have presented an algorithm to select a subset of points to optimally approximate a digital ink curve. In particular, it is able to find an approximation with a specified number of points and providing the minimum cumulative error or to select the minimum number of points required to satisfy a given error threshold. The algorithm is based on dynamic programming and is independent of the choice of compatible error function, and we have examined its performance with three: the Arc-Chord Length

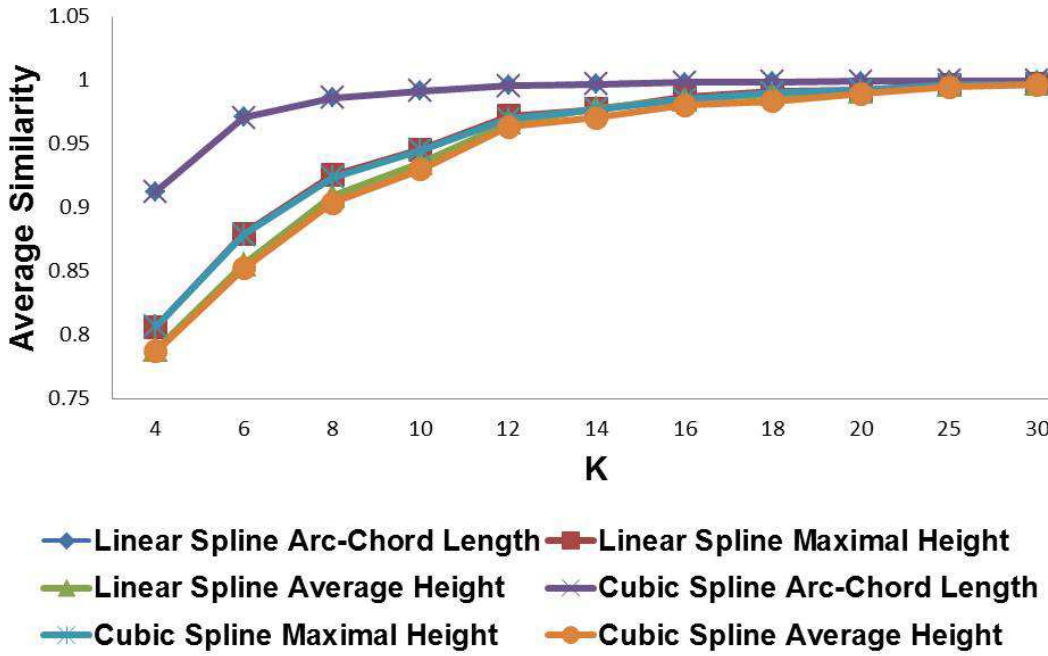


Figure 2.6: The average similarity between the approximated curve and original curve.

Error, the Maximal Height Error, and the Average Height Error. These were chosen for their computational efficiency and natural geometric meaning. Our experiments have shown that the Arc-Chord Length Error outperforms the others in terms of average time cost in both linear and cubic spline cases.

There are a few interesting directions we would like to pursue in the future. First, in addition to the cumulative error, we would like to restrict the error on each segment. This would allow control of the local deviation as well as the global deviation. Second, we wish to perform measurements with more types of error, including those involving differences in the spatial derivatives.

Chapter 3

Flexible Representation of Sophisticated Digital Ink

Modern devices have reached maturity to support different drawing and writing activities, e.g. diagramming and digital painting, by providing additional information such as pen tilt angle, pen tip force, timestamp, *etc.* However, few digital ink formats allow recording these additional information. In this chapter, we investigate the suitability of InkMl to represent sophisticated digital ink. InkML is a W3C standard format. It provides application-defined channels to support sophisticated digital ink representation. To demonstrate our ideas, we present two virtual brush models, Round Brush and Tear Drop Brush. Both models can be used to improve the rendering quality of digital ink. The Tear Drop Brush has been adopted into InkML at our suggestion. This chapter is based on my master thesis [24].

Different drawing and writing activities require different rendering brushes. For example, diagramming may best be done with a brush that behaves like a pen or pencil, while digital painting and East Asian calligraphy may require a bristled brush which can result more stroke variations. To accommodate these needs, we present two virtual brush models. Both models can be used to improve the rendering quality of digital ink.

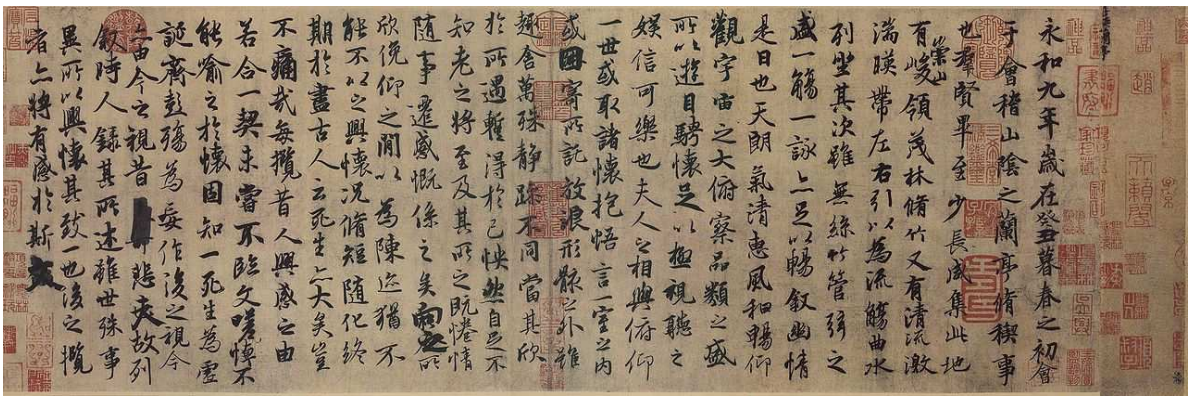


Figure 3.1: An example of Chinese calligraphy. The Orchid Pavilion Gathering by Xizhi Wang, a well known calligrapher [25].

3.1 Introduction

Although digital ink technologies have evolved over years, the "brushes" used by digital ink applications are still simpler than real brushes. These brushes are normally two dimensional, only recording X and Y coordinates. Even in sophisticated applications, brush tip shapes tend to be at most ellipses or rectangles with a major and minor dimension and perhaps an angle. Few of them provide support to record additional information such as pen tilt angle, pen tip force, timestamp, *etc.*, which are essential when simulating more complex writing instruments. With the widespread application of pen-based devices, it has become increasingly popular to render digital ink strokes with different styles. This is particularly useful when aesthetic and decorative effects are desired. Many Asian writing systems use a round brush with long bristles as the writing implement. While the elliptical or rectangular contact shapes may be sufficient to model writing with pens, they cannot represent the characteristics of a physical brush. Take Chinese calligraphy as an example, as shown in Figure 3.1, since the brush is made of soft bristles, there are abundant width variations. The width variations even exist in a single stroke. For instance, in Figure 3.2, the stroke is fat at the beginning as writers pressed hard when the brush tip dropped on the paper. Then it become thinner while the brush was moving. As soon as the brush reached the end of the stroke, the tip suddenly turned back and the stroke got fat again due to the delay of the long brush tail.

Various brush models have been developed and used on computers to simulate bristled brush properties. One of the earliest attempts is that of Steve Strassmann

Figure 3.2: A simple stroke of Chinese calligraphy.

[26]. He described an investigation into a realistic model of painting and proposed a method which can be used to simulate the properties of wet paint brush. Wong and Ip [27] presented an method to simulate the physical process of brush stroke creation. This method uses a parameterized model which captures the brush's 3D geometric parameters, the brush bristle properties as well as the variations of ink deposition along a stroke trajectory.

Our work is similar to those in that we all keep track of each ink point and calculate its contour to simulate brush properties. Our work is also apart of those as we are using InkML, a W3C standard, to represent and render digital ink.

In this chapter, we present two virtual brush models, *Round Brush* and *Tear Drop Brush*, to render digital ink in different styles. The Round Brush may best be used for diagramming while the Tear Drop Brush may be suitable for digital painting or East Asian Calligraphy. Both brush models have been integrated into InkChat [28], a whiteboard application which allows conducting multi-party collaboration.

The remainder of this chapter is organized as follows. In Section 3.2, we explain how to represent sophisticated digital ink using InkML. Section 3.3 presents the brush models. In Section 3.4, we summarize the chapter with a brief discussion and suggest a few directions of future work.

3.2 InkML Representation

After considering various alternatives including Unipen [4], JOT [6], ISF [29], and SVG [30], we arrived at a design using InkML. InkML is a W3C standard format. It provides application-defined channels to support sophisticated ink representation.

Each application-defined channel provides a coordinate value at each ink point. For example, the InkML point (x, y, r, l, θ) specifies a five-channel trace format. Applications may arbitrarily define x, y to indicate the position of an ink point, r, l to measure the brush head radius and the tail length, θ to represent the rotation angle of the brush tail. Such feature is extremely useful as it allows applications to flexibly represent digital ink strokes and to render them in different styles according to the context.

Taking advantage of InkML, we propose a method to render sophisticated digital ink in different styles. We first capture digital ink from pen-based devices and retrieve information such as location, pen tip pressure, timestamp and tilt angle. To render it with a particular brush property, we can simulate the dynamic brush shapes by calculating the contour at each ink point.

To demonstrate our ideas, we have developed two virtual brush models.

3.3 Brush Models

Calligraphy cannot be rendered using simple lines or curves due to the stroke width variations. We thus use a different approach: we calculate the dynamic brush shape at each point while the brush is moving. Obviously, different brushes result in different 2D contours. Even using the same brush, different pressure may cause contour size and shape variations. Therefore, after capturing the digital ink from pen-based device, we convert it to brush-specific parameters and then render it according to the brush type.

The first brush we developed is the Round Brush. We have also developed a more sophisticated one, Tear Drop Brush, which can be used to simulate Chinese calligraphy. The following two sub-sections illustrate the two brush models, respectively.

3.3.1 Round Brush

The Round Brush, as its name suggests, draws each ink point as a filled circle. We use three parameters to model the Round Brush, as shown in Figure 3.3. The x and y are to indicate the position of the ink point and the r is to measure the circle radius

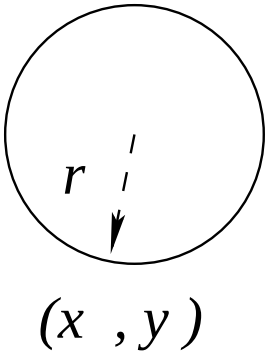


Figure 3.3: The model of Round Brush

which is a function of pen tip pressure. In general, the harder you press the brush, the larger the circle. In our model, the radius of the circle r is linearly proportional to the pressure p, i.e.

$$r = kp,$$

where k is a heuristic constant.

Pen-based devices work under certain sampling rates. A higher sampling rate indicates it can sample more ink points in a unit time and therefore leads to a greater chance that two successive circles would overlap. A lower sampling rate indicates it can sample fewer ink points in a unit time and leads to a smaller chance that two successive circles would overlap. In addition, if the brush moves fast, the chance of overlapping would correspondingly decrease. Practically two successive circles do not overlap if the brush is moving. Even if they are overlapped, there is a great chance that they are not fully overlapped. As long as full overlap does not exit, there will be a gap between each pair of successive circles. In order to obtain a smooth rendering, we fill the gaps by calculating the external tangents and coloring the area determined by the four tangent points, as shown in Figure 3.4.

3.3.2 Tear Drop Brush

The idea of the Tear Drop Brush comes from the Round Brush and it is more sophisticated. Holding a brush upright and pressing the tip on to paper, then lifting it up quickly, would leave a round dot. If you then press the tip on to paper and drag it for a tiny distance, there would be a dot that looks like a tear drop left on the paper. However, if you press the brush very hard, the soft bristles would spread

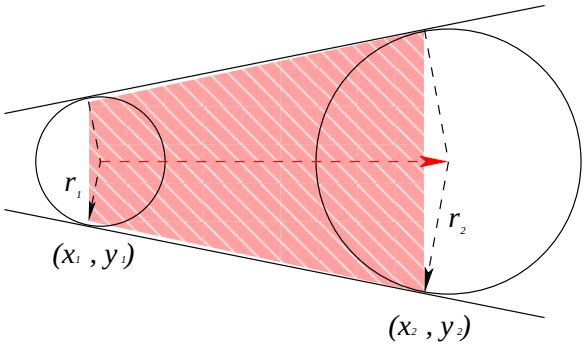


Figure 3.4: Filling the gap between two successive Round Brush ink points.

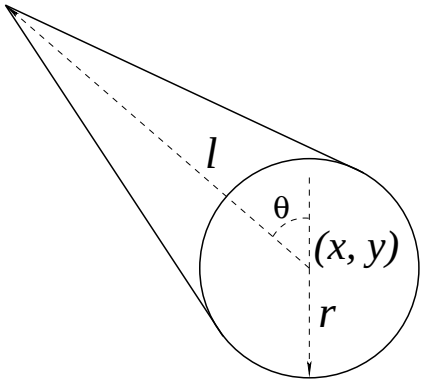


Figure 3.5: The model of Tear Drop Brush

out and the ink mark would become a fat round dot again. Based on this idea, we use five parameters to model the Tear Drop Brush, as shown in Figure 3.5. x and y indicate the position of the ink point, r represents the head radius, θ indicates the rotation angle of the tail, and l measures the tail length, the distance from the circle centre to the tail end.

In principle, the head radius should be a function of pressure. The harder you press the brush, the larger the radius would be. In our approach, we set the radius r to be directly proportional to the pressure p, i.e.

$$r = kp,$$

where k is a heuristic constant.

The tail length l ranges from r to L which is the length of the brush bristles. Its

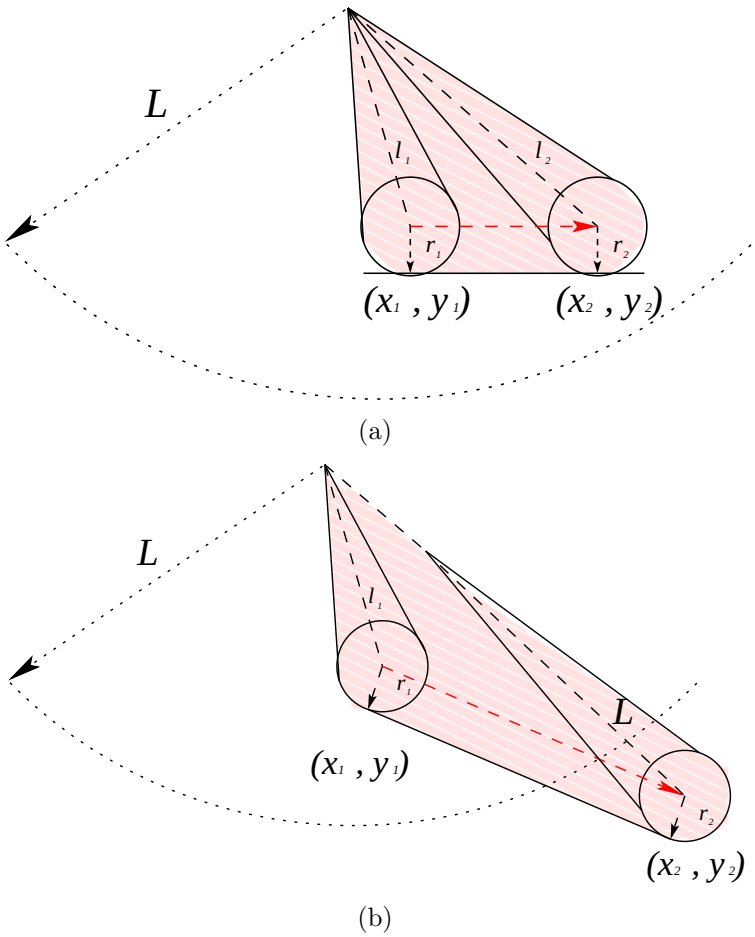


Figure 3.6: Tear Drop Brush. (a) The tail end remains. (b) The tail end moves.

value depends on the distance between old tail end and the position of the new ink point.

We can describe this using what is called the "donkey wagon" model. Imagine the tail end of the tear drop shape is a wagon on a rope being pulled by a donkey at the center of the head. When the donkey changes direction the wagon follows the rope, which will be a straight line to the donkey's current position. If the donkey moves in a direction that makes the rope lose, then the wagon stays in the same place. Likewise, there are two cases when moving the Tear Drop Brush. If the distance between the old tail end and the new ink point is smaller than L, the new tail end would stay at its original place and we fill the area as shown in Figure 3.6(a). If the distance exceeds L, the new tail end would move to somewhere on the line determined by the old tail end and the new ink point, while the tail length becomes L. And we fill the area as shown in Figure 3.6(b)

Figure 3.7: An example of Chinese calligraphy using Tear Drop Brush.

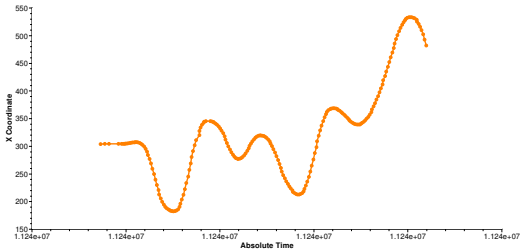
In the implementation, we retrieve coordinates and pen tip pressure at each ink point from a Wacom tablet running on Ubuntu 8.10. We convert it to the Tear Drop Brush parameters and then save them into InkML points (x, y, r, θ, l). An example image of the Tear Drop Brush calligraphy can be seen in Figure 3.7. Figure 3.8 shows the plot of the first stroke of Figure 3.7.

3.4 Summary

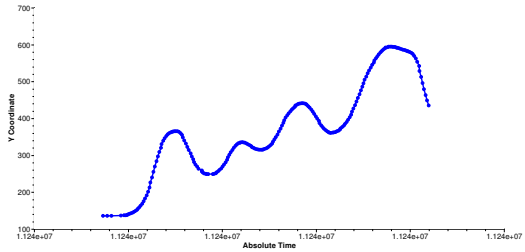
Our work is an attempt to render digital ink with calligraphic properties. We have found a simple mathematical model to be sufficient to represent writing with a round, bristled brush. Our model is able to portray a good approximation to the contact shape of a brush being dragged along the writing surface, depositing ink with changes in pressure and direction. This "tear drop" contact shape has been adopted into InkML at our suggestion.

The main utility of our model is that it can be used to construct a good approximation to the time history of a bush contact shape, and hence the stroke outline, from the$(x, y,$ pressure) history of a single contact point. This allows a digital pen to be used to form realistic strokes, as though created by a paint brush or writing brush. In particular, this allows basic calligraphic brush writing using digital ink.

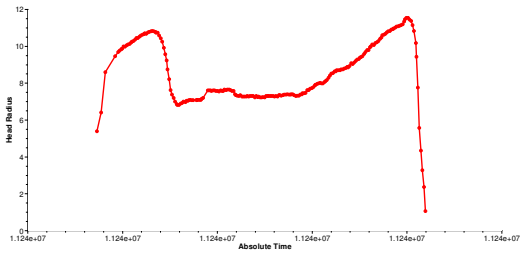
(a) The ink stroke to plot (in red).



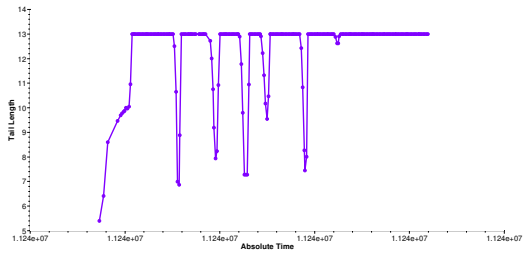
(b) The plot of x coordinates



(c) The plot of y coordinates



(d) The plot of the head radius



(e) The plot of the tail length

Figure 3.8: The plot of Tear Drop Brush parameters

Chapter 4

Portable Representation of Digital Ink

The MIT Sketch Markup Language (SketchML) and the Ink Markup Language (InkML) are both used for sketch data representation. Techniques for exchanging sketch data between the two formats, however, are currently not readily available. In this article, we present a data binding solution for two-way conversion between SketchML and InkML. We show how to transform SketchML files to InkML archiving and streaming style. This allows sketch data to be used in collaborative environments where real-time sharing is desired. In the reverse conversion, we bind InkML elements to SketchML elements. This makes sketch data that is represented by InkML available to existing SketchML applications. We have tested these ideas in a shared whiteboard application and found them to perform well. This chapter is based on the article "From MIT SketchML to InkML, or There and Back Again" [31] co-authored with Stephen M. Watt, that presented at 10th IAPR International Workshop on Document Analysis Systems.

4.1 Introduction

While once somewhat exotic and specialized, pen and touch enabled devices are now ubiquitous. It is therefore becoming increasingly important to be able to exchange drawing and writing input between platforms and applications. Various vendors record pen or touch input either in their own proprietary formats, or simply as images

that lose important information. Various formats for digital ink have been proposed earlier, including Unipen and JOT, but these have either been restricted to special uses or have not received wide acceptance. Presently two data formats stand out as sufficiently general for wide-spread use: *InkML* and *MIT SketchML* (we will refer to as SketchML in the following text for simplicity). Both of these are XML-based data formats, similar in that they both record trace information, but otherwise quite different. This chapter examines the question of what is involved in converting between these formats. Before describing the technical challenges and our solutions, we first set the stage with some basic ideas.

InkML (Ink Markup Language) is an open and up-to-date recommendation standard which is proposed by the W3C (World Wide Web Consortium). It can be used by digital ink applications for multiple purposes. n particular, it supports digital ink sharing based on the concept of context [32]. The context is represented by the <context> element which contains various aspects of real-time scenario, including <canvas>, <canvasTransform>, <traceFormat>, <inkSource>, <brush> and <timestamp>, which in turn specifies the canvas, canvas transformation, trace format, ink source properties, brush properties and time stamp.

SketchML (Sketch Markup Language) is an XML-based open format which was developed by the Design Rationale Group at MIT. It provides a collection of elements to represent user's sketch as well as other meta information such as the study and the domain that the sketch was created for. It enables stroke grouping and annotation by allowing for the creation of the <shape> element. SketchML is useful in sketch data representation and annotation [33]. However, it lacks support for sketch sharing which is critical in many collaborative scenarios. These scenarios, such as classroom teaching, distance education, work-group meeting, and collaborative document annotation, typically involve multiple participants, which is complicated compared with the single user case. The application in these scenarios could be even more restricted when the collaborative environment is heterogeneous. The restrictions stem not only from differences between operating systems and platforms but also from the differences in characteristics of the pen devices, such as channel properties, screen resolution and so on.

We are interested in exploiting the benefits of both the InkML and the SketchML formats so that sketch data can be flexibly represented as well as easily shared. This demands a technique for efficient conversion between the two formats. However, such

Metadata Element	Description
<sketcher>	The author
<mediaInfo>	The referenced external file, e.g. audio file
<study>	The study that the sketch was created for
<domain>	The domain that the sketch was created for

Sketch Data Element	Description
<point>	An individual ink point.
<shape>	A shape consisting of a sequence of points or smaller shapes. It may also contain other metadata such as color, pen tip, raster and so on.

Figure 4.1: SketchML elements.

conversion is not straightforward. The organizations of the elements within the two formats are different. Hence, they impose different requirements for markup processors and generators and give different computational complexities for certain operations. In SketchML, contextual information are delivered by the attributes of the <shape> element which represents a series of strokes. All the ink points associated with the strokes are given by references to the <point> elements which are typically stored separately from the <shape> element. This is opposite to the InkML format. In InkML, contextual information are retrieved from strokes and stored apart. Depending on scenarios, the contextual elements may appear either within <definitions> element or prior to <trace>/<traceGroup> elements. References to the contextual elements are made by the <contextRef> and the <brushRef> attributes of the <trace>/<traceGroup> element. The <trace>/<traceGroup> element carries explicit coordinate values instead of references to ink points. Therefore, this conversion problem is to transform the SketchML format to the InkML format and vice versa in efficient ways. In an abstract view, this transformation can be seen as semantically binding elements between the two formats.

In this chapter, we present a technique to interchange sketch data between the SketchML format and the InkML format. By converting from SketchML to InkML, we make SketchML data acceptable by digital ink applications that support InkML. We also exploit the InkML streaming features so that sketch can be shared in real time between participants, who may be in the same room or across the planet. By converting from InkML to SketchML, we make sketch data that is represented by InkML compatible to existing SketchML applications.

The rest of this chapter is organized as follows. In section 4.2, we discuss the

```
<annotationXML>
  <sketcher>
     <id>86bf5a7b-b71b-4912-a3aa-e686f5abdf1b</id>
     <dpi x="96" y="96" />
  </sketcher>
</annotationXML>
```

Listing 4.1: SketchML <sketcher> as InkML annotation

technique to convert sketch data from the SketchML format to the InkML format. In section 4.3, we discuss the technique of the reverse conversion, from the InkML format to the SketchML format. In section 4.4, we present a sketching interface implementation to demonstrate our ideas. In section 4.5, we summarize with a brief discussion.

4.2 SketchML to InkML

SketchML can be used by applications to store and manipulate sketch data. The <sketch> element is the root element of any SketchML file instances. It contains a set of metadata elements as well as sketch data elements. Figure 4.1 shows each of them.

The metadata elements includes <sketcher>, <mediaInfo>, <study> and <domain>, which in turn specifies the person who created the sketch, the referenced external file (e.g. audio file), and the study and the domain that the sketch was created for. The sketch data elements, including <point> and <shape>, are used to represent a series of strokes. Each stroke consists of a sequence of contiguous ink points. To support stroke grouping and annotation, multiple strokes are allowed to exist within a named <shape> element. The <shape> element may also contain other metadata such as color, pen tip, raster and so on. The UUIDs that are assigned to each point in SkethcML are encoded as additional channels in the InkML trace format.

To convert from the SketchML format to the InkML format, the metadata elements can be easily represented by InkML <annotationXML> elements. An example of using InkML <annotationXML> element to represent SketchML metadata element is shown in Listing 4.1.

InkML provides two styles of sketch data generation and usage — archiving and streaming. The two are semantically equivalent and can be faithfully converted from one to the other [34]. They are designed for different application scenarios. The archiving style provides more direct support for operations such as search and retrieval, whereas the streaming style provides more direct support for instantly streaming sketch data with lower overhead on the wire. Therefore, the sketch data elements in SketchML must be converted accordingly. We will discuss the conversion to each style in detail.

4.2.1 SketchML to Archival InkML

InkML archiving typically handles strokes that have been collected over some span of time and may re-organize them so that preferably the strokes be state-free. Therefore, the associated contextual information can be stored apart from the strokes. It is usually represented by the <context> elements. Each of them is assigned an identifier using the `xml:id` attribute. References to these contextual elements are made using the `contextRef` attributes of each stroke.

To accommodate the InkML archiving style, we represent each SketchML <shape> element by a pair of an InkML <traceGroup> element and a <context> element. The <traceGroup> contains the shape's coordinates. The <context> element contains contextual information. Reference to the <context> element is made by the `contextRef` attribute of the <traceGroup> element.

If the <shape> element contains sub-stroke (i.e. a reference to another <shape> element), a child <traceGroup> element will be created. Otherwise, an InkML <trace> element will be created to store coordinate values of associated ink points. An InkML <traceFormat> element will be created at the same time to describe the format that is used to encode each ink point within the <trace> element. In particular, it defines the sequence of coordinate values that occurs within the <trace> element. The order of declaration of channels in the <traceFormat> element determines the order of appearance of the coordinate values within the <trace> element. An example of conversion from SketchML to InkML archiving style is shown in Listing 4.2.

Conversion from SketchML to InkML archiving is useful in that it makes SketchML data available to InkML applications. By retrieving contextual information from

```
<?xml version="1.0" encoding="UTF-8"?>
<ink xmlns="http://www.w3.org/2003/InkML">
 <definitions>
  ....
  <context traceFormatRef="#TF0" xml:id="Ctx2">
   <brush xml:id="Brush2">
    <brushProperty name="transparency" value="255" />
    <brushProperty name="rasterOp" value="copyPen" />
    <brushProperty name="color" value="#000000" />
    <brushProperty name="tip" value="ellipse" />
    <brushProperty name="laysInk" value="true" />
   </brush>
   <timestamp time="1180113658602" xml:id="TS2"/>
  </context>
  <traceFormat xml:id="TF0">
   <channel name="X" orientation="+ve" type="decimal"/>
   <channel name="Y" orientation="+ve" type="decimal"/>
   <channel name="TIME" orientation="+ve" type="integer"/>
   <channel name="MUID" orientation="+ve" type="integer"/>
   <channel name="LUID" orientation="+ve" type="integer"/>
   <intermittentChannels>
    <channel name="P" orientation="+ve" type="decimal"/>
   </intermittentChannels>
  </traceFormat>
  ...
 </definitions>
 ...
 <traceGroup contextRef="#Ctx2" xml:id="TG2">
  <annotationXML>
   <shape id="5a6b5e88-5274-404e-abb2-19c2e40351c9"
          author="86bf5a7b-b71b-4912-a3aa-e686f5abdf1b"
          height="1" name="stroke"
          type="SubStroke" width="150"/>
  </annotationXML>
  <trace xml:id="Trace2">
   3305.0 9966.0 1180113657038 2758944186448628176
   -6377473126013613401 23.0,
   3330.0 9952.0 1180113657046 -7998112319767163665
   -5372399576976323103 29.0
  </trace>
 </traceGroup>
 ...
</ink>
```

Listing 4.2: Example of conversion from SketchML to InkML archiving stlye. The `MUID` and `LUID` represent the most and least significant bits of the UUID of each point, respectively.

strokes, we can also make certain contextual elements in InkML reusable by other strokes and thus could potentially optimize sketch interpretations in the future.

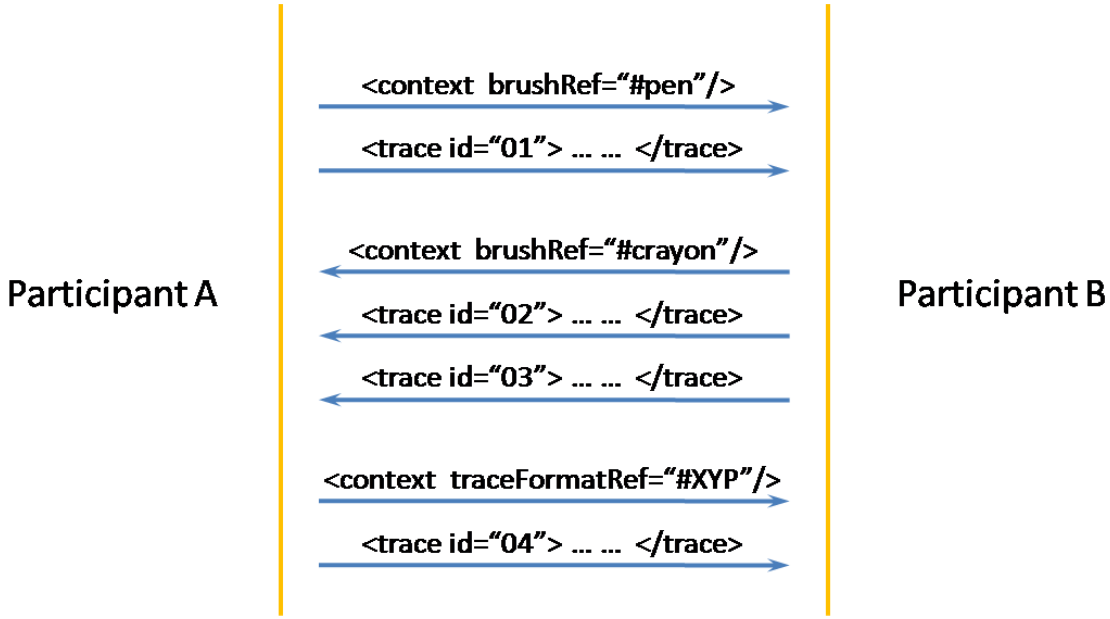


Figure 4.2: Example of InkML Streaming style

4.2.2 SketchML to Streaming InkML

InkML streaming delivers strokes in sequential time order. Compared with InkML archiving, contextual information are inserted into the stream of the strokes, as needed, to provide interpretation. Changes to the current context are given by <context> elements. This corresponds to an event-driven model of ink generation, where events which result in contextual changes map directly to elements in the markup. An example of the InkML streaming style is shown in Figure 4.2. In this example, the participant A first sends a context change event. It notifies the participant B that the stroke "01" must be interpreted as a "pen" stroke. The participant B then sends another context change event followed by stroke "02" and stroke "03". It specifies that both the strokes must be interpreted as "crayon" strokes. Later, the participant A switched trace format to "XYP". It notifies the participant B that each ink point of the stroke "04" should be treated as (x, y, p) triplet, where the x and y are the position coordinates of the ink point, and the p is the pen tip pressure.

To convert sketch data from the SketchML format to the InkML format, an InkML <traceGroup> element along with a <context> element will be generated for each SketchML <shape> element. To accommodate the streaming style, the <context>

```
<context traceFormatRef="TF0" xml:id="Ctx2">
   <brush xml:id="Brush2">
    <brushProperty name="transparency" value="255" />
        ...
   </brush>
</context>
<traceGroup contextRef="#Ctx2" xml:id="TG2">
  <annotationXML>
    <shape id="5a6b5e88-5274-404e-abb2-19c2e40351c9"
      type="SubStroke" width="150"
      author="86bf5a7b-b71b-4912-a3aa-e686f5abdf1b"
      height="1" name="stroke"/>
  </annotationXML>
  <trace xml:id="Trace2">
    3305.0 9966.0 1180113657038
      2758944186448628176 -6377473126013613401 23.0,
    3330.0 9952.0 1180113657046
      -7998112319767163665 -5372399576976323103 29.0,
    ...
  </trace>
</traceGroup>
```

Listing 4.3: Example of conversion from SketchML to InkML streaming style.

element must appear prior to the <traceGroup> element. An example of conversion from SketchML to InkML streaming style is shown in Listing 4.3.

Converting from SketchML to InkML streaming style is useful in that it allows SketchML data to be shared instantly as it was captured. This enables SketchML applications to be used in collaborative scenarios, such as classroom teaching, distance education, work-group meeting, and collaborative document annotation, where sketch data must be transmitted in real time and properly interpreted by other participants.

4.3 InkML to SketchML

Conversion from InkML to SketchML is simply a reverse process. All the meta data elements, the <edit> element and the <speech> element can be easily restored from the corresponding <annotationXML> elements in InkML. Each stroke and its associated context will be altogether converted to a SketchML <shape> element. If the stroke needs to be transformed (i.e. its associated context contains any InkML <mapping> elements), the generated <shape> element will reflect such transformation. All of its ink points or sub-strokes will be transformed accordingly.

4.4 Implementation and Experiments

To test our ideas we have developed a software implementation, InkChat (`http://www.orcca.on.ca/InkChat/`). InkChat is a cross-platform whiteboard application which allows conducting and archiving collaborative sessions that involve synchronized voice and sketch on a shared canvas. It accepts both the InkML and the SketchML formats and incorporates the technique presented for conversion. In particular, it allows any SketchML elements to be processed in the InkML streaming style so that it can be transmitted to other participants in real time.

4.5 Summary

We have shown how to convert from SketchML to InkML archiving style and to InkML streaming style, each supports certain operations more directly. By converting to InkML archiving style, we can make SketchML file available to InkML applications. By converting to InkML streaming style, we can enable SketchML applications to be used in collaborative environments where sketch can be seen by other participants in real time. We have also shown how to conduct the reverse conversion, from InkML to SketchML. It makes sketch data that is represented by InkML compatible to existing SketchML applications. As a proof-of-concept, we have presented InkChat, a collaborative whiteboard application. It incorporates the conversion technique and shows good performance.

Part II

Recognition

Chapter 5

Determining Points on Handwritten Mathematical Symbols

In a variety of applications, such as handwritten mathematics and diagram labelling, it is common to have symbols of many different sizes in use and for the writing not to follow simple baselines. In order to understand the scale and relative positioning of individual characters, it is necessary to identify the location of certain expected features. These are typically identified by particular points in the symbols, for example, the baseline of a lower case "p" would be identified by the lowest part of the bowl, ignoring the descender. We investigate how to find these special points automatically so they may be used in a number of problems, such as improving two-dimensional mathematical recognition and in handwriting neatening, while preserving the original style. This chapter is based on the article "Determining Points on Handwritten Mathematical Symbols" [35] co-authored with Stephen M. Watt, that appeared in the proceedings of 2013 Conferences on Intelligent Computer Mathematics.

5.1 Introduction

Many digital ink applications allow handwritten characters in various sizes and in different locations. For example, in mathematics, subscripts and superscripts appear relatively smaller than normal text and are written slightly below or above it.

Moreover, these subscripts and superscripts may themselves have subscripts or superscripts. Such notation is easily read and understood. This involves determining the relative baselines and sizes of symbols. This process may present various ambiguities, for example whether a particular symbol is a lower case "p" or an upper case "P" giving the subscripted p_q or the juxtaposed Pq.

5.2 Determining Points

In order to find the scale and offset of individual characters, it is necessary to identify the location of certain expected features which are typically defined by particular points. These particular points occur at different locations in different symbols, and the precise location can vary in different handwriting samples of the same symbol. For example, the baseline of lowercase "p" would be identified by the lowest part of the bowl, ignoring the descender. In contrast, the baseline of lowercase "k" would be identified by the toes. In this chapter we refer to a point such as this, that determines the height of a metric line, as a *determining point*. Knowing the determining points of each symbol can help us solve a number of problems. For example, one can use the determining points to improve two-dimensional mathematical recognition. By comparing the baseline locations and the sizes of adjacent symbols, one can identify subscripts and superscripts (e.g. S_2, $S2$, S^2) with more confidence. Another application is in handwriting neatening. Since handwritten symbols often come with variations in alignment and size, certain transformations based on determining points can be applied to obtain normalized samples while preserving the original writing style.

5.3 Challenges

Recording determining points for an individual handwritten symbol is easy. One can manually annotate the symbol with the positions of all its determining points. However, finding determining points for all symbols in a collection is much more challenging. First, with a large database the labour for manual annotation would be prohibitively costly. Secondly, applications such as mathematics involve a large variety of symbols derived from a range of alphabets and other sources. In practice,

many of them are often poorly written and there is no fixed dictionary of words to aid in disambiguation [11]. This increases the difficulty to find determining points reliably. Meanwhile, each person's handwriting is unique — even identical twins write differently [12]. Even if a training database were to be fully annotated, it is not entirely clear how this should best be used to identify the points of interest in new input. Last, but not least, the usual methods for detecting determining points depend on device resolution significantly. With rapidly evolving technology, this means that new algorithms cannot use archival data directly and therefore must be "re-sampled" (interpolated).

5.4 Previous Work

We are interested in the problem of how to automatically find determining points of handwritten mathematical symbols and to use them in a variety of problems. Considerable related work has been conducted, some of which we highlight here. Pechwitz and Märgner [36] proposed an algorithm that can find determining points from symbol skeleton approximated by piecewise linear curve. However, these determining points are only useful in detecting baseline locations. In 2010, Infante Velázquez [37] developed an annotation tool to record determining points manually for handwritten characters represented in InkML. The determining points were later used to neaten new handwriting, making it uniform in size, alignment and slant while preserving writers' particular writing styles. However, this tool recorded each determining point with absolute coordinates and was therefore subject to device resolution and variations in style. As device resolution may vary among different vendors and over generations of technology, this approach is not device-independent. Similar problems exist in [38]. In addition, Zanibbi *et al.* [39] proposed a technique to automatically improve the legibility of handwriting by gradually translating and scaling individual symbol to closely approximate their relative positions and sizes in a corresponding typeset version. This technique detects baseline locations by comparing symbols' bounding boxes, which leads to troubles with vertical placement and scale. For example, it fails to distinguish between "$x2$" and "x^2". In 2012, Hu and Watt [17] presented an algorithm to find turning points that determine the shape of characters, but that approach lacked the ability to capture the geometric meaning of each determining point and therefore does not provide sufficient information to calculate certain desired symbol metrics, such as the location of baseline. Harouni *et al.* [40] later pro-

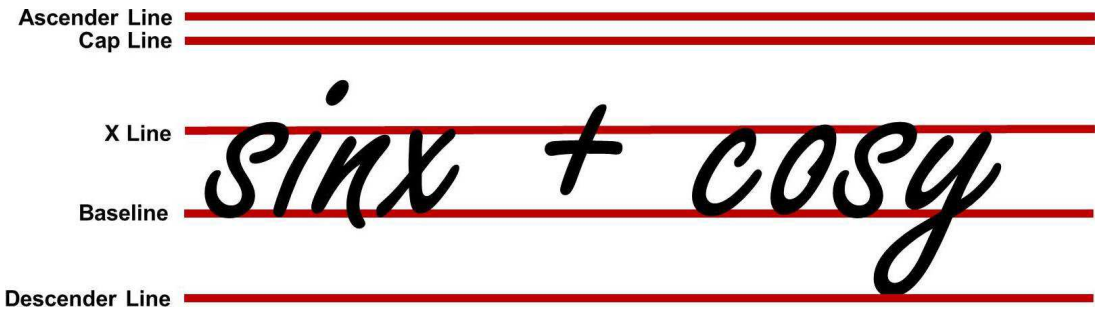


Figure 5.1: An example to illustrate the concepts of metric lines.

posed a method to find determining points in handwritten Arabic characters. The method consisted of two stages. In the first stage, the raw input data were converted to a standard format using smoothing, normalization and interpolation techniques. In the second stage, each stroke of input characters was split into several pieces. The method calculated the local maximum and minimum of each piece and recorded them as determining points. However, this method is not optimal as it requires extra effort to split strokes and may generate undesired determining points that lack meaning.

5.5 Objectives

In this chapter, we present an algorithm to find determining points automatically and suggest how they may be used in areas such as improving two-dimensional mathematical recognition and in neatening handwriting. The basic approach is to identify the points of interest on one average instance of each type of symbol, and to use this information to find the corresponding points on newly written symbols. We borrow ideas from typography, where a number of determining points are identified to measure the metrics of different font families, and apply these to handwriting. We consider several types of determining points, which, in turn, determine certain metrics. These include the locations of the five main metric lines, i.e. the baseline, x line, ascender line, cap line, and descender line, as shown in Figure 5.1, as well as symbol width and slant. To make the determining points device-independent, the algorithm first converts all handwritten symbols into parametric curves approximated by truncated orthogonal series, mapping each symbol to a single point in a low dimensional vector space of series coefficients. We then compute the average symbol for each class by computing the average of the points for the class in the vector space. The determining points of interest are identified on these average symbols. From these, the algorithm

(a) (b)

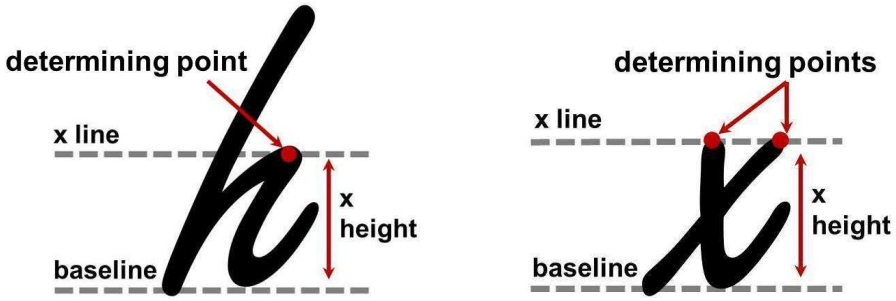
Figure 5.2: Baseline with (a) one determining point (b) multiple determining points.

can derive corresponding determining points in samples automatically. The beauty of this algorithm is that it is writer-independent. We only need to annotate once, on the average symbols. This reduces cost significantly. Furthermore, the algorithm is device-independent as all symbols are represented in the functional space, which is robust against changes in device resolution.

The remainder of this chapter is organized as follows. Section 5.6 discusses several types of determining points that are useful in finding symbol alignment lines. In Section 5.7, we present the algorithm that can identify determining points in handwritten mathematical symbols automatically. Section 5.8 evaluates the performance of the algorithm. We then investigate the possible use of the algorithm in a number of problems in Section 5.9. Section 5.10 concludes the chapter.

5.6 Handwriting Metrics

In order to understand the scale of individual symbols, it is necessary to identify the location of certain expected features which are typically defined by a number of determining points. These determining points have locations that vary from symbol to symbol, but typically occur where parts of the symbols touch certain invisible horizontal lines. To discuss this, we adopt concepts from typeface design. In this chapter, we consider several types of determining points related to the following metrics. We concentrate on symbols used in European alphabets. Many other writing systems would have other metric lines determined in a similar way.



(a) (b)

Figure 5.3: X line and X height with (a) one, and (b) multiple determining points.

Baseline

Most scripts share the notion of *baseline*. It is a guide line for writing so that adjacent symbols can retain their horizontal alignment. It is also used as the reference to obtain other metrics such as x height, ascender height, etc. While some symbols such as lower case "p" may extend below the baseline, it serves as the imaginary base for most symbols. Figure 5.2 shows examples of baselines and their determining points. As shown in Figure 5.2(b), the three legs of the lowercase "m" are not completely aligned. In such case, multiple determining points are identified and the location of the baseline may be determined by the average y value of all the determining points.

X Line and Height

The *x line* falls at the top of most lowercase symbols, such as "a" and "y", and is located over the baseline. Some symbols may extend above the x line, such as "h" where the x line is located at the top of the shoulder. The *x height* is the distance between the baseline and the x line. Figure 5.3 shows an example of x line and associated determining points. Certain symbols, such as lowercase "x", may have multiple determining points to define the x line. In such a case, the location of the x line is determined by the average of their y values.

Ascender Line and Height

The part of a lowercase symbol, such as "h" and "k", that extends above the x line is known as an *ascender*. The *ascender line* is located above the x line and

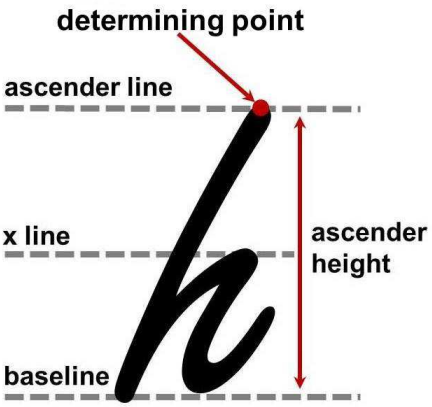


Figure 5.4: Ascender line and height

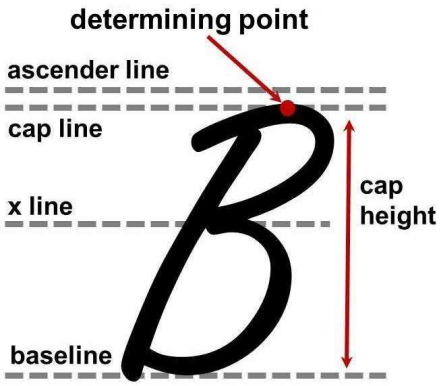


Figure 5.5: Cap line and height

is determined by the height of the ascenders. The *ascender height* is the distance between the baseline and the ascender line. Figure 5.4 shows an example of an ascender line and ascender height. The location of the ascender line is determined by the determining point shown in red. In the case that there are multiple determining points, the location of the ascender line is given by the average y value of all the relevant determining points.

Cap Line and Height

The *cap line* is usually located below the ascender line, but is not limited to that position. It is used to measure the height of uppercase symbols, which is the distance between the baseline and the cap line. Figure 5.5 shows an example of cap line and *cap height*. The location of the cap line is determined by the determining point shown in red. In the case that there are multiple determining points, the location of the cap line is determined by the average y value of all the determining points.

Descender Line and Height

The *descender line* is located below the baseline. It is used to align descenders, which are the parts of symbols that extend below the baseline. Figure 5.6 shows an example of a descender line and descender height. If there are multiple determining points, the location of the descender line is given by the average y value of all the determining points.

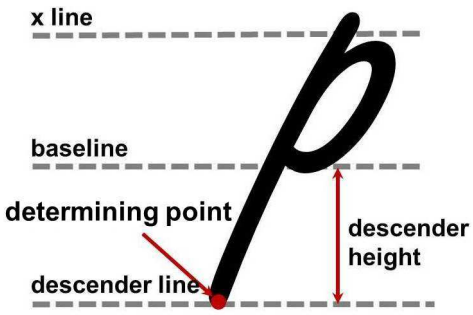


Figure 5.6: Descender line and height

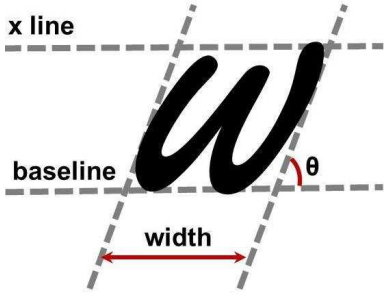


Figure 5.7: Slant (θ) and width

Slant and Width

In some handwriting styles, symbols are written with inclination either to the left or to the right. The degree of inclination is referred to as the *slant*. The *width* of a symbol is given by the horizontal distance from the left-bounding and right-bounding lines with the given slant. Figure 5.7 shows an example of symbol width and slant.

5.7 Algorithms

In this section, we present an algorithm to find automatically the determining points for newly written symbols. The algorithm derives determining points for a new symbol from the known determining points of an annotated average symbol of the same type.

Average Symbols

We classify symbols so that symbols that are written the same way and could be interpreted the same way are in the same class. So, for example, there may be several classes for the numeral "8", depending on whether the symbol is written with one continuous stroke or two separate strokes, which stroke is written first and the direction of writing. On the other hand, a Latin letter "O" and the numeral "0" could belong to the same class.

Taking each sample as a point in the functional approximation space, it has been found in earlier work that classes of points are almost completely pair-wise separable by single hyperplanes. Thus the convex hulls of the class point sets are to a good

(a) (b)

Figure 5.8: (a) Samples provided by different writers. (b) The average symbol.

approximation non-intersecting. Any point on a line segment between two sample points of the same class falls within that class. It is therefore meaningful to compute the average of a set of known samples for a class as the average point in the function space

$$\bar{C} = \sum_{i=1}^{n} C_i/n,$$

where n is the number of the samples and C_i is the coefficient vector for the i^{th} sample. Figure 5.8(a) shows a set of samples provided by different writers and Figure 5.8(b) shows the average symbol.

Deriving Determining Points from Average Symbols

Our algorithm is based on the observation that the average symbols typically look similar to the samples of the same class. Within a given class, the features present in one sample should be present in other samples and at a similar location. We can take the location to be the arc length along the ink trace to the defining point of the feature. We assume that, if two symbols are sufficiently similar, the locations of corresponding determining points will be similar (given by distance along the curve).

This suggests that we can find the determining points of a new symbol by taking the known locations on an annotated symbol and making an adjustment. In more detail, to detect the determining points in a sample, we start with an annotated sample in the same class. For now, this will be the average of the training samples, annotated with its determining points. Each annotation consists of the location (as arc length), the type of determining point (e.g. baseline, x line, etc) and whether it is located at a local minimum or local maximum of y value.

Algorithm 3: LocateDeterminingPoints

Input : A, the coefficient vector of a reference symbol.

$D_A = [(s_1, T_1, K_1), \ldots (s_n, T_n, K_n)]$, a vector of determining points. For each, the position is given as arc length s_i on the curve of A, the value T_i states which type of metric line is being defined, and the value K_i states whether the metric line is given by a local minimum or local maximum at $y_A(s_i)$.

S, the coefficient vector for the input sample whose determining points are to be found.

Output: $D_S = [(\ell_1, T_1, K_1), \ldots, (\ell_n, T_n, K_n)]$, giving the locations, ℓ_i, and types of the determining points of S.

1. Let $x_A(s)$, $y_A(s)$, $x_S(s)$, $y_S(s)$ be the coordinate functions of the symbols given by A and S.
2. **for** $i \in 1..n$ **do**
   - **if** $K_i =$ ***max*** **then**
     - $f \longleftarrow -y_S$
   - **if** $K_i =$ ***min*** **then**
     - $f \longleftarrow +y_S$
   - $\ell_i \longleftarrow s$ such that $f(s)$ is minimized near s_i.
   - *Note this is the local minimum of a real univariate polynomial and any standard method may be used. For example, we use Newton's method to solve* $f'(s) = 0$ *with initial point* $s = s_i$.
3. Return$[(\ell_1, T_1, K_1), \ldots, (\ell_n, T_n, K_n)]$

For each determining point of the annotated sample, we guess that the corresponding determining point on the new sample will be near the same arc length location. So we take the point at that location in the new sample and follow the trace upward or downward, depending on whether that determining point is supposed to be at a local minimum or local maximum. This can be easily done using a number of numerical methods. In our implementation, we applied Newton's method to solve $y'(s) = 0$. A formal algorithm is given in Algorithm 3.

Figure 5.9 shows examples of using Algorithm 3. Figure 5.9(a) shows the determining points annotated on the average symbol "η". This is the reference symbol A in the algorithm. Figures 5.9(b1) and 5.9(c1) show two example input samples S with initial approximate locations s_i for the determining points. Figures 5.9(b2)

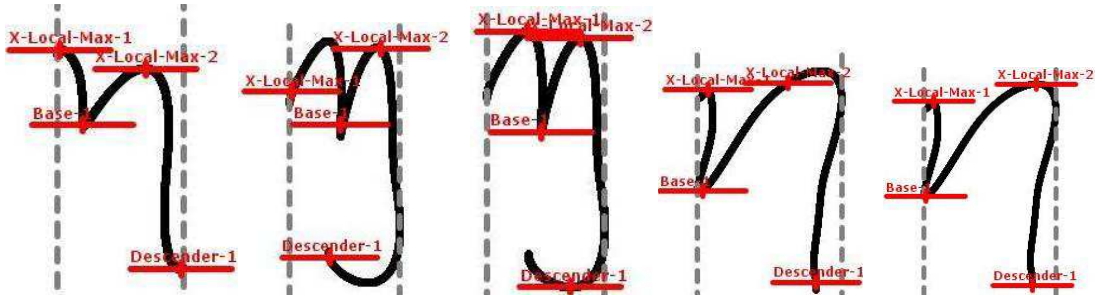


Figure 5.9: Automatically finding determining points. (a) Average symbol "η". (b1) Sample 1 initial approximations and (b2) with determining points found. (c1) Sample 2 initial approximations and (c2) with determining points found.

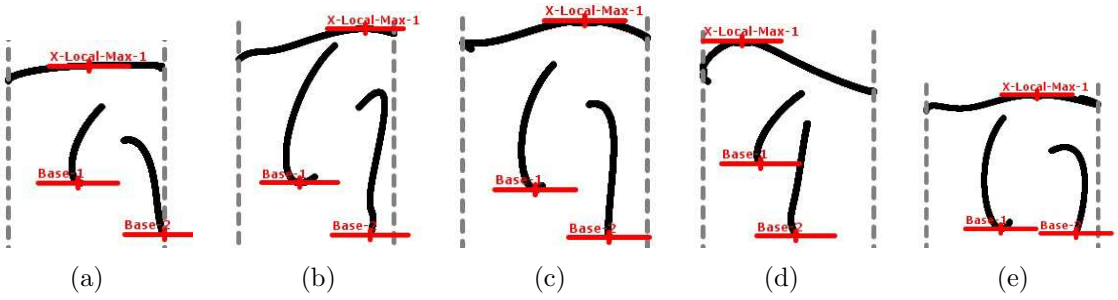


Figure 5.10: Automatically finding determining points. (a) Average symbol "π". (b-e) Determining points derived from the average symbol.

and 5.9(c2) show the determining points found at locations ℓ_i. Figure 5.10 shows several examples of determining points found for samples of "π".

5.8 Experiments and Testing

We developed a software tool to annotate handwriting samples with their determining points. Figure 5.11 shows the user interface. By selecting a nearby location, the tool is able to find the target determining point automatically. The locations of all the metric lines discussed in Section 5.6 can be detected. Multiple determining points may exist for certain metrics lines. In such circumstances, the location of the corresponding metric line is determined by the average of the values given by all the determining points of that kind. Symbol slant can also be recorded by adjusting a spinner. Symbol width is automatically detected with slant considered.

To evaluate the performance, we have tested the algorithm against a large hand-

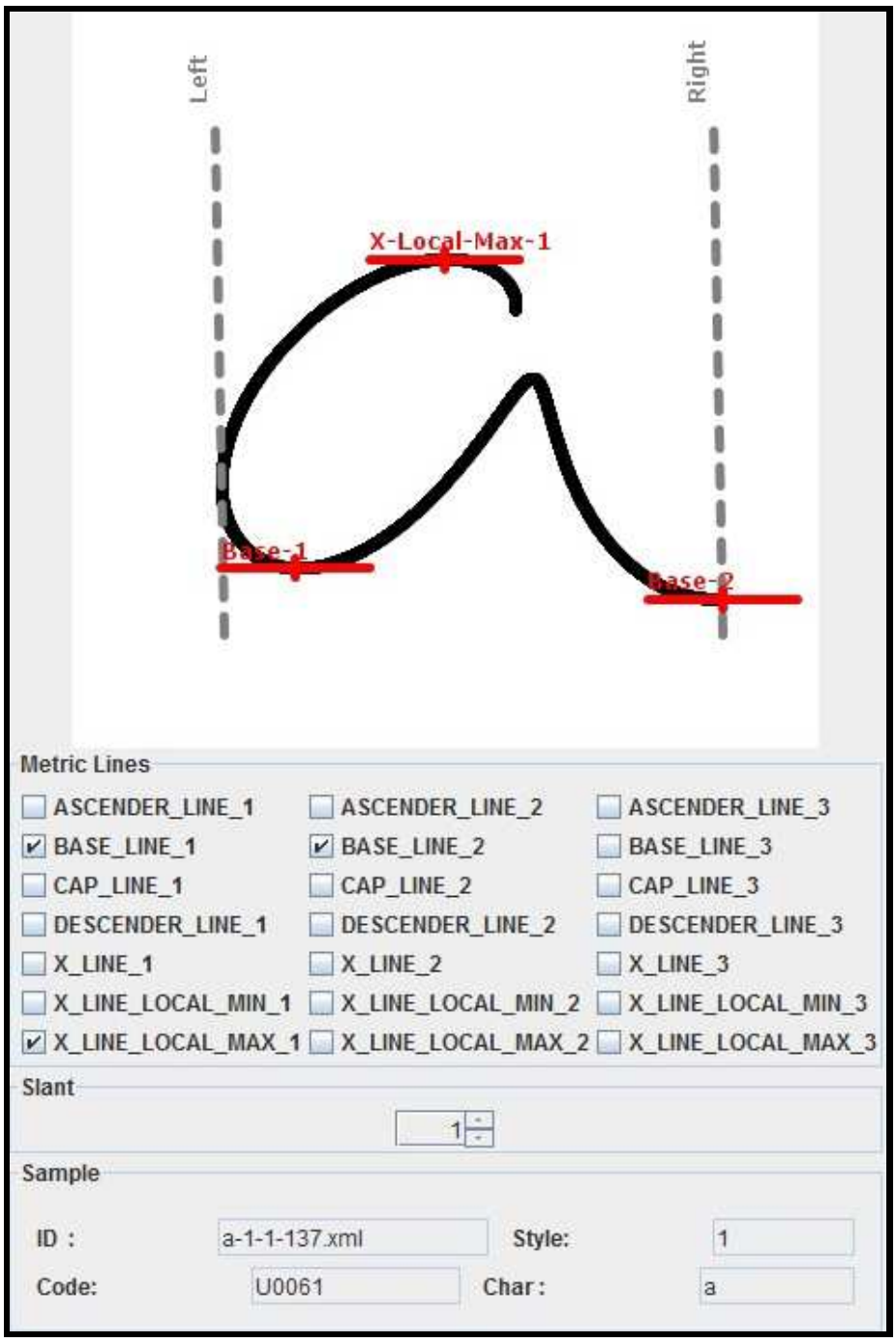
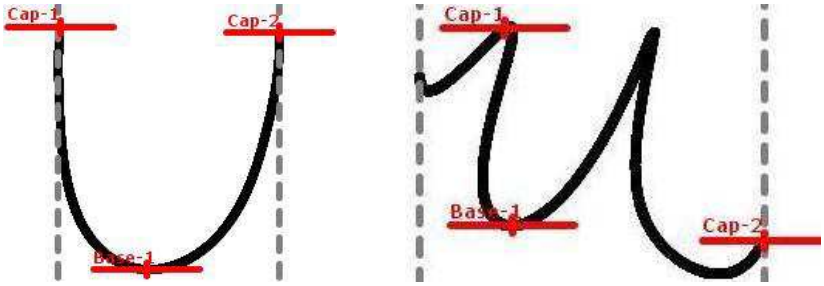


Figure 5.11: Software tool to annotate a symbol with determining points.

writing dataset. The handwriting dataset we used contained altogether 64944 samples of 240 different symbols. Most of the samples are Latin and Greek letters, digits, operators, or other mathematical symbols provided by various writers. All of these samples had been classified in advance. As some symbols were written in different styles (e.g. completely different forms, different numbers of strokes, or strokes in dif-



(a) (b)

Figure 5.12: Failure example: (a) average symbol, (b) target with one point misplaced.

ferent orders), a total of 382 classes were examined. We first computed the average symbol for each class, in which determining points were identified using the software tool shown in Figure 5.11.

We then computed determining points for all the samples using Algorithm 3. The number of determining points varied from 2 to 5, according to the sample. If any of the determining points were mis-positioned, we considered it as incorrect. We chose up to 30 samples randomly from each class and examined their correctness visually. In total, we examined 8119 samples, of which 421 samples have at least one mis-positioned determining point. This gave a measured error rate of 5.2%.

We found the error was introduced mainly from two sources. The first was misclassified samples in the original data set. These were either mis-labelled (e.g. "e" of style 1, instead of "e" of style 2), or had strokes given in a different order from the usual. In this latter case, we have the option of defining a new style or normalizing the order of the strokes. The second source was that some samples are significantly different from the average symbol. As a result, the determining points in the average symbol may not be mapped correctly to those dissimilar samples.

As misclassified samples were errors in the training data, rather than errors by the algorithm, we excluded those samples from the experiment. We further added 39 new classes (giving 421 classes in total) to split out those samples with different stroke orders. After these corrections, the measured error rate decreased to 2.0% (9593 samples reviewed, of which 189 samples had at least one mis-positioned determining point).

To address the second issue, that of points mis-positioned because the sample was far from the average shape, we used a homotopy [41] between the average and the

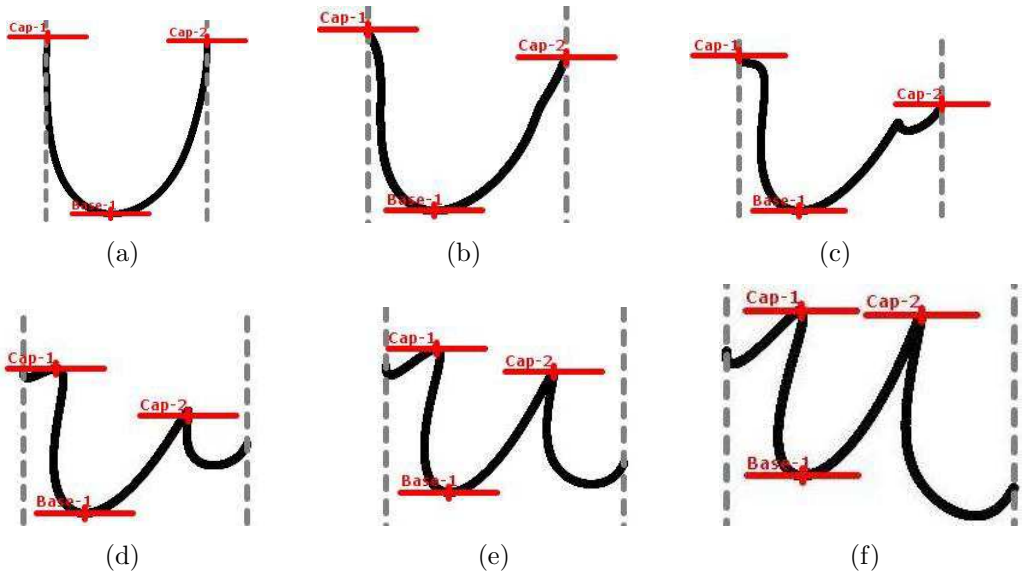


Figure 5.13: Success in 5 steps: (a) average symbol (b) step 1 (c) step 2 (d) step 3 (e) step 4 (f) step 5 = target.

test sample in a multi-step method. Recall that, in the function space, a line from the average symbol to the test sample lies entirely within the class. By dividing this line into several equal steps, we may apply Algorithm 3 several times to follow the determining points through the homotopy. If $\bar{C}$ is the average symbol for the class and C_{targ} is the input sample, then the line joining the two points in the function space is given by $C(t) = (1-t)\bar{C}+tC_{targ}$, with t ranging from 0 to 1. The determining points should move smoothly as the character is deformed by the homotopy, and we can choose a step size. Figure 5.12 shows an example where Algorithm 3 fails to identify one of the determining points when applied naively. However, when applied in a 5 step homotopy, it succeeded, as shown in Figure 5.13.

We have tested the multi-step method against the same handwriting dataset. We chose up to 30 samples randomly from each class and examined their correctness visually. The measured error rates are reported in Figure 5.14. The samples that failed in the 10-Step and 20-Step methods typically either had slants that interfered with the strategy of using local minimum or maximum y value to find determining points or that were very badly written. For these samples, our algorithm was able to identify some determining points correctly but not all of them, as shown in Figure 5.15. Note that the points found would in any case be sufficient for most applications.

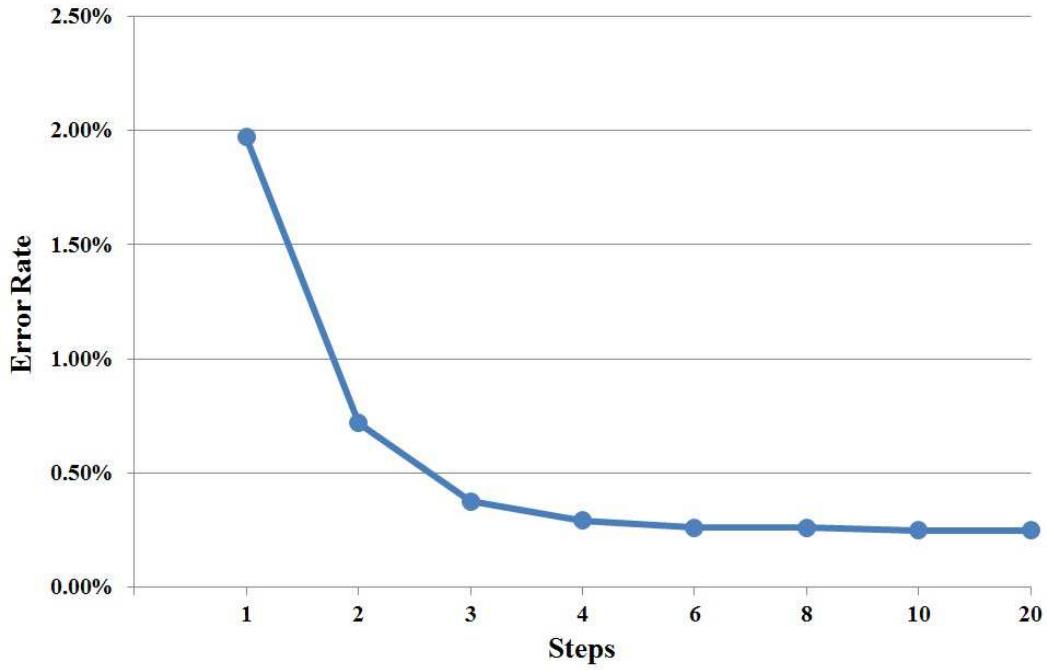
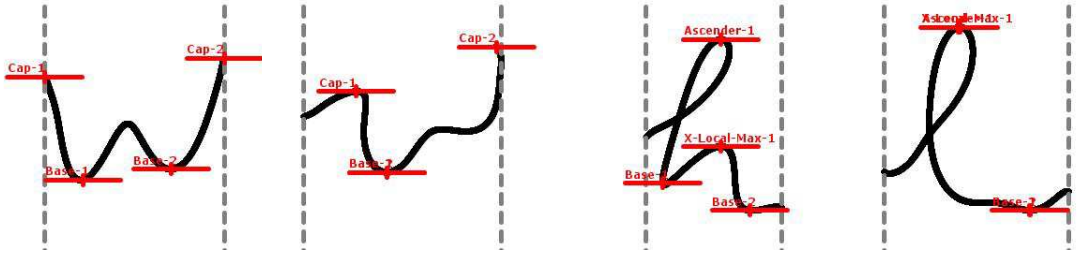


Figure 5.14: Error rates of the multi-step method on 9593 samples.



(a) (b) (c) (d)

Figure 5.15: Multi-step failures: (a) Average, (b) target. (c) Average, (d) target.

5.9 Use Cases

Determining points can be used in a variety of digital ink applications to solve different problems. Here we describe two scenarios in which determining points have been found useful.

(a) (b) (c) (d) (e) (f)

Figure 5.16: Juxtaposition ambiguity.

Handwriting Recognition

Juxtaposition ambiguity is common in mathematical handwriting recognition. This is usually caused by symbols that are next to each other are written in different sizes and at different heights. Figure 5.16 shows an example with several relative positionings of two characters. The first character can in each case be a "P" or "p" and the second can be interpreted as a "q" or "9". Together there could be a variety of possible interpretations:

$$\begin{array}{cccccc} P^9 & P9 & P_9 & & p^9 & p9 & p_9 \\ P^q & Pq & P_q & & p^q & pq & p_q \end{array}$$

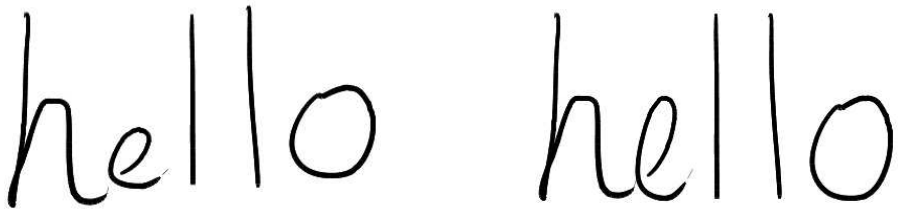
However, by comparing symbols' baseline locations and sizes, we can predict each expression with more confidence. This is because the baselines of subscripts and superscripts are typically placed slightly below or above the normal line of text and their sizes are relatively smaller. Note that to determine the relative position, it is definitely not sufficient to compare the baselines of the symbol bounding boxes. This is seen in Figure 5.17(d). Similarly, having an imputed baseline determined by symbol class (such as at 50% height for "q") is insufficient. We thus find it is important to find and use the symbol's determining points.

Handwriting Neatening

Handwriting neatening is becoming possible in some digital ink applications. It is used to transform handwriting to obtain visually appealing output while preserving the original writing style. Figure 5.18 shows an example. By identifying the determining points of each character, we can shift and scale these characters to make corresponding

(a) (b) (c) (d)

Figure 5.17: Disambiguation by baselines. (a) P9 (b) Pq (c) pq (d) p9



(a) (b)

Figure 5.18: Neatening using determining points. (a) original, (b) neatened.

metrics lines aligned properly, as shown in Figure 5.18(b). Figure 5.19 shows a second example. In this case, all characters including the superscripts and subscripts were adjusted in order to obtain a normalized output. Transforming the function $y(s)$ for each symbol is the simplest approach to neatening. A more aggressive approach is to replace each input symbol with the appropriately scaled version of the average of like symbols seen by the same writer, and further transformations can be employed. However, this is beyond the scope of the present chapter.

5.10 Summary

We have presented an algorithm to identify automatically the determining points in handwritten symbols. Identifying these determining points helps us better understand the scale of individual characters as well as find the locations of certain desired features. In contrast to existing methods, which treat digital ink traces as a collection of discrete points, this algorithm relies on interpreting ink traces as single points in a functional space. This allows device independence, on one hand, and a simple formulation of homotopic deformation, on the other.

(a) (b)

Figure 5.19: Neatening using determining points. (a) original, (b) neatened.

Various features can be recorded by using the determining point algorithm. The nature of the determining points depends on the symbol set used. In our case, the symbols were based mostly on those of European languages and mathematical operators, so the baseline, x line, ascender line, descender line and cap line were used.

To evaluate the performance of the algorithm, we have tested it against a database of handwritten mathematical characters. The experiments showed promising results. To demonstrate possible use of determining points, we have described two scenarios: handwriting recognition and handwriting neatening, in both of which determining points have been found useful.

There are a few directions we would like to pursue in the future. First, we wish to include determining points in our handwriting recognizer. It is expected that, combined with ambient baseline information, this will improve the recognition rate. Secondly, we would like to investigate using rotation- and slant-invariant techniques [42, 43] in conjunction with the present methods. At a more detailed-level, we would like to annotate all samples in our database using a supervised multi-step method. This will allow us to perform a more satisfying statistical analysis of the effectiveness of our method. Finally, before incorporating these techniques in our recognition framework, we would like to investigate the correlation between the model-sample distance and the number of steps required for low error rates, and how the number of required steps varies by class.

We would like to thank Isaac Watt for helping to organize the handwriting dataset used in the experiments.

Chapter 6

Identifying Features via Homotopy on Handwritten Mathematical Symbols

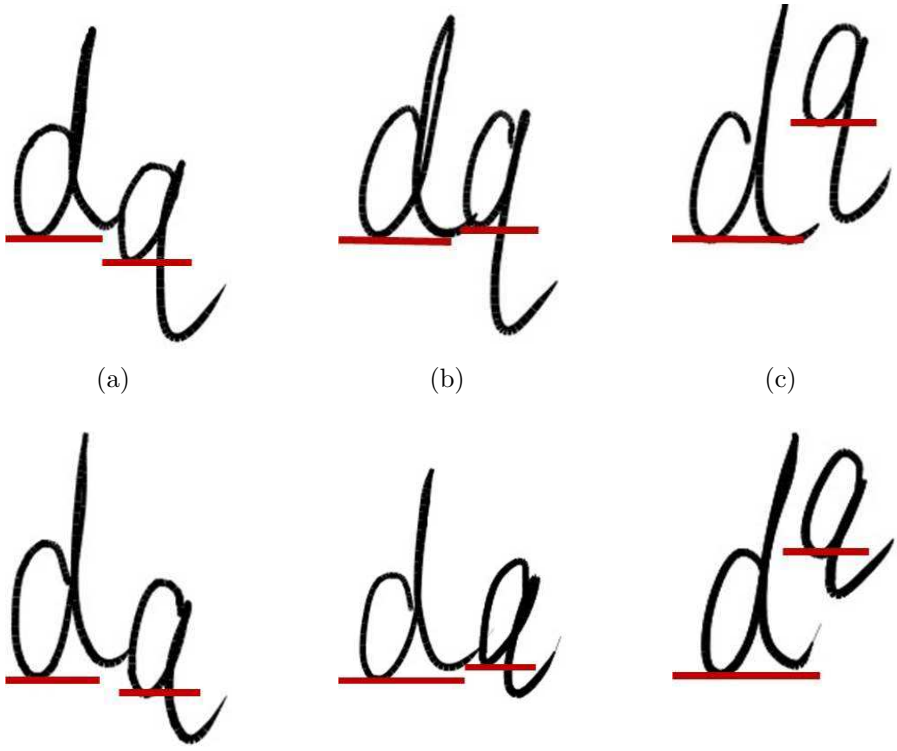
In handwritten mathematics, it is common to have characters in various sizes and for writing not to follow simple baselines. For example, subscripts and superscripts appear relatively smaller than normal text and are written slightly below or above it. Rather than use the location, features and size to identify the character, it may be more effective to do the reverse — to use knowledge about specific characters to determine baseline, size, etc. In this approach, it is necessary to find the location of certain expected features that are determined by particular points. In earlier work, we have presented a method to derive the determining points for a new instance of a symbol from those on an average model for each symbol type. For those characters that are significantly different from the average instance, one can use a numerical homotopy between the average instance and the target character, and apply the determining point algorithm at each step. The present chapter studies the factors to be taken into account in performing such homotopies. We examine two strategies for possible starting points for the homotopy, and we examine the relation between the distance and the number of steps required. The first starting point strategy performs a homotopy from the average of samples of the same type. The second strategy uses a homotopy from the nearest neighbour with known determining points. Our experimental results show a useful relation between the homotopy distance and the number of steps usually required and improved strategies to find determining points

for poorly written characters. This chapter is based on the article "Identifying Features via Homotopy on Handwritten Mathematical Symbols" [44] co-authored with Stephen M. Watt, that appeared in the proceedings of 2013 International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

6.1 Introduction

Mathematical handwriting recognition differs from natural language handwriting recognition in many ways [45, 46, 47]. Symbols are taken from many alphabets, for example, and are written in a two dimensional layout in various sizes with layout and size carrying meaning [48]. The vocabulary of different symbols is larger than in western alphabets, and there are more distinct types of strokes than in East Asian ideographs. In addition, there is no fixed dictionary of words to help with disambiguation [11]. On the other hand, characters tend to be well separated. One of the problems arising from these differences is that baseline estimation is more difficult and may not be used reliably to disambiguate characters. Subscripts and superscripts appear relatively smaller than normal text and are written slightly below or above it. Moreover, these subscripts and superscripts may themselves have subscripts or superscripts. Such notation makes the analysis of the spatial relationships between symbols challenging as it introduces various ambiguities. For example, whether a particular symbol is a lower case "p" or an upper case "P" makes the difference between a subscripted p_q or the juxtaposed Pq.

In Chapter 5, we have explored the idea that it may be more desirable to identify the possible local baselines in a formula, as well as other metric lines and sizes, from features on individual symbols, rather than the other way around. The features that can be used to do this are typically determined by points appearing at symbol-dependent locations. For example, if a symbol is a lower case "p", then the baseline is determined by the lowest point in the bowl (loop), but if it is an upper case "P", then the baseline is determined by the lowest point of the stem. In this chapter, we refer to such a point, one that determines the height of a metric line, as a *determining point*. Knowing the locations of the determining points can help us identify the size and spatial relationships of symbols and consequently use them in formula recognition. For example, one can use the determining points to help resolve the juxtaposition ambiguity problem which commonly exists in mathematical handwriting recognition.



(a) (b) (c)

(d) (e) (f)

Figure 6.1: Baselines: (a) d_q (b) dq (c) d^q (d) d_q (e) dq (f) d^q

This problem arises when symbols that are next to each other are written in different sizes and at different height. Figure 5.16 shows an example. Note that to determine the relative position, it is definitely not sufficient to compare the baselines of the symbols' bounding boxes. This is clearly seen Figure 6.1(c). Similarly, having an imputed baseline determined by symbols (such as at 50% height for "q") would mistreat either Figure 6.1(b) or 6.1(e). We thus find it is important to locate and use the symbol's determining points. Figure 5.17 shows another example of the juxtaposition problem caused by variance of baselines.

We have previously addressed the problem of how to find determining points automatically [35]. Our approach was to represent symbols as approximating polynomial curves, $(x(s), y(s))$, parameterized by arc length and to specify determining points by their type and arc length location on a model character. The determining points on new samples are then identified by finding local minimization or maximization of $y(s)$ using the model points' location as an initial estimate. The previous chapter showed that, for characters that are significantly different from the average instance, one can

use a numerical homotopy between the model instance and the target character, and apply the local extremum-finding algorithm at each step.

The present chapter extends the earlier work by addressing several questions related to the choice of homotopy method and the efficiency implications of those choices. We concentrate on two strategies: The first performs a homotopy from the computed and hand-annotated average symbol the class. The second uses a homotopy from the nearest neighbour in the class with known determining points, the determining points of the nearest neighbour having been derived automatically from other labelled points of the same class.

The questions we ask are the following:

- To find the determining points of a new sample, what are the differences between starting the homotopy from the average symbol of the class versus starting from the nearest labelled neighbour?
- What is the relationship between the distance from the new sample to the model point and the number of homotopy steps required?
- How can we best use the results to find the determining points in new samples?

The remainder of this chapter is organized as follows. In Section 6.2 we describe related work. Section 6.3 presents the algorithm to identify determining points using homotopy strategies. Section 6.4 presents the experimental results and suggests improved strategies to find determining points in poorly written characters. In Section 6.5 we summarize the chapter.

6.2 Related Work

Several related problems have been studied in the past. One of the early attempts is the projection method [49, 50], which accepted binary images as input and counted the black pixels line by line. The baseline was then identified by finding the line with the maximum number of black pixels. The method is not entirely reliable. There are cases where the method fails to estimate the baseline locations. This is a concern for our application where there is often only one or a few characters with the same local baseline.

A more advanced method was presented by Pechwitz and Märgner [36] in 2002, based on polygonally approximated symbol skeleton. The method was able to extract certain features from skeletons and then estimate the location of baselines. However, this method is limited to a specific language and cannot be used to detect other metric lines.

In 2010, Infante Velázquez [37] developed a tool to locate determining points in handwritten characters represented in InkML through manual annotation. This tool allowed recording each determining point with absolute coordinates. As the sampling rate and resolution vary between different vendors and over generations of technology, such representation is, however, not device-independent. Similar problems exist in [38].

Zanibbi *et al.* [39] proposed a technique that can gradually translate and scale individual symbols to closely approximate their relative positions and sizes in a corresponding typeset formula. As this technique used bounding boxes to detect baseline locations, it may lead to troubles with vertical placement and scale. For example, it fails to distinguish between "$x2$" and "x^2". Similar problems exist in [51, 52, 53].

Hu and Watt [17] later described an algorithm that can find those special points that determine the shape of a character. But that approach was unable to capture the geometric meaning of each determining point and therefore did not provide sufficient information to calculate desired symbol metrics, such as the baseline location.

Harouni *et al.* [40] presented a method to find determining points in handwritten Arabic characters. It first divided each ink stroke into pieces, in each of which the local extremum was computed. The points that achieved the local extremum were later defined as the determining points. However, this method is not suitable for mathematics in that it requires extra effort to split strokes and may generate undesired determining points that lack geometric meaning.

6.3 Algorithm

In Section 1.2, we have explained how to represent handwritten samples using coefficients of a functional approximation. This representation is device-independent, allowing us to focus on finding determining points without worrying about device-

dependency. It also enables various symbolic-numeric polynomial algorithms to conduct useful analysis on the handwritten mathematical samples.

Our algorithm to find determining points on a sample is based on the observation that samples of the same class typically have the same features. We can specify the location of a determining point by the value of the normalized arc length parameter at which it occurs on a model symbol. Although the precise locations of the determining points on a new sample will be different, we expect them to occur near these parameter values. To find precise locations of the determining points on the new sample, we can follow the ink trace, starting from the approximate locations, until local vertical extrema of the right type (minimum or maximum) is found.

More specifically, to detect the determining points of samples, we first choose a reference symbol for each class. We then annotate the reference symbol, identifying the locations of determining points of interest. As the computation is based on curves $(x(s)), y(s))$ parameterized by normalized arc length s, the locations of the determining points are recorded by their s values in the interval $[0, 1]$. We can then compute determining points in target samples, starting from the location of the corresponding determining points in the reference symbol. Each determining point of the sample can then be identified by finding the extremum of the polynomial $y(s)$ near the starting point. This can be achieved using any one of a number of numerical methods. In our implementation, we applied Newtons method to solve $y'(s) = 0$. These steps are shown in Algorithm 3, which is similar to that of [35]. The difference is that article always uses the average instance of each class as the reference symbol, but here we allow other suitable reference symbols. This allows us to investigate the effect of choosing different starting points.

6.3.1 The Reference Symbols

We examine two types of reference symbols as the choice of starting point for the homotopy: the *average symbol* and the *nearest neighbour*.

Average Symbol: The functional representation of digital ink traces has the advantage that the curves become points in a linear space. It therefore makes sense to talk about the average of several points. Our first choice for reference symbol is the

average symbol of a class, computed as

$$C_{avg} = \sum_{i=1}^{n} C_i/n,$$

where n is the number of the samples and C_i is the coefficient vector for the i^{th} sample. Note that this computed average is in general not actually one of the sample points. Figure 5.8(a) shows a small set of samples provided by different writers and Figure 5.8(b) shows the average symbol.

Choosing the average symbol as the reference symbol for a class is an intuitive choice because presumably it has little deformation compared to other samples in the class. Moreover, it will lie within the convex hull of the set of points being averaged. Earlier work has shown [54] that, with this representation at relatively low approximation degree, the symbol classes are almost completely pairwise separable by single hyperplanes. We may therefore take the volume enclosed by the convex hull of points as being properly included within the class. Finally, the choice of average symbol is attractive in that only one symbol per class need be annotated.

Nearest Neighbour: Another choice of starting point for the homotopy is the nearest neighbour in the class of the new symbol. This has the advantage that it should resemble the new symbol more closely than the average symbol. We would therefore suspect this starting point might require fewer homotopy steps. This choice of starting point is not as simple as the average symbol, however, because all of the symbols in the database must be labelled with their determining points. For a database of any size, this is not something that can be done by hand.

6.3.2 Homotopy

For those samples that are significantly different from the reference symbol, we use a numerical homotopy between the reference symbol and the target sample and trace the movement of the determining points. Because the classes are linearly separable in the function space a line from the reference symbol to the target sample lies entirely within the class. Dividing this line into several equal steps, we may apply Algorithm 3 at each step to follow the determining points through the homotopy. If C_{ref} is the reference symbol for the class and C_{targ} is the new sample target, then the line joining

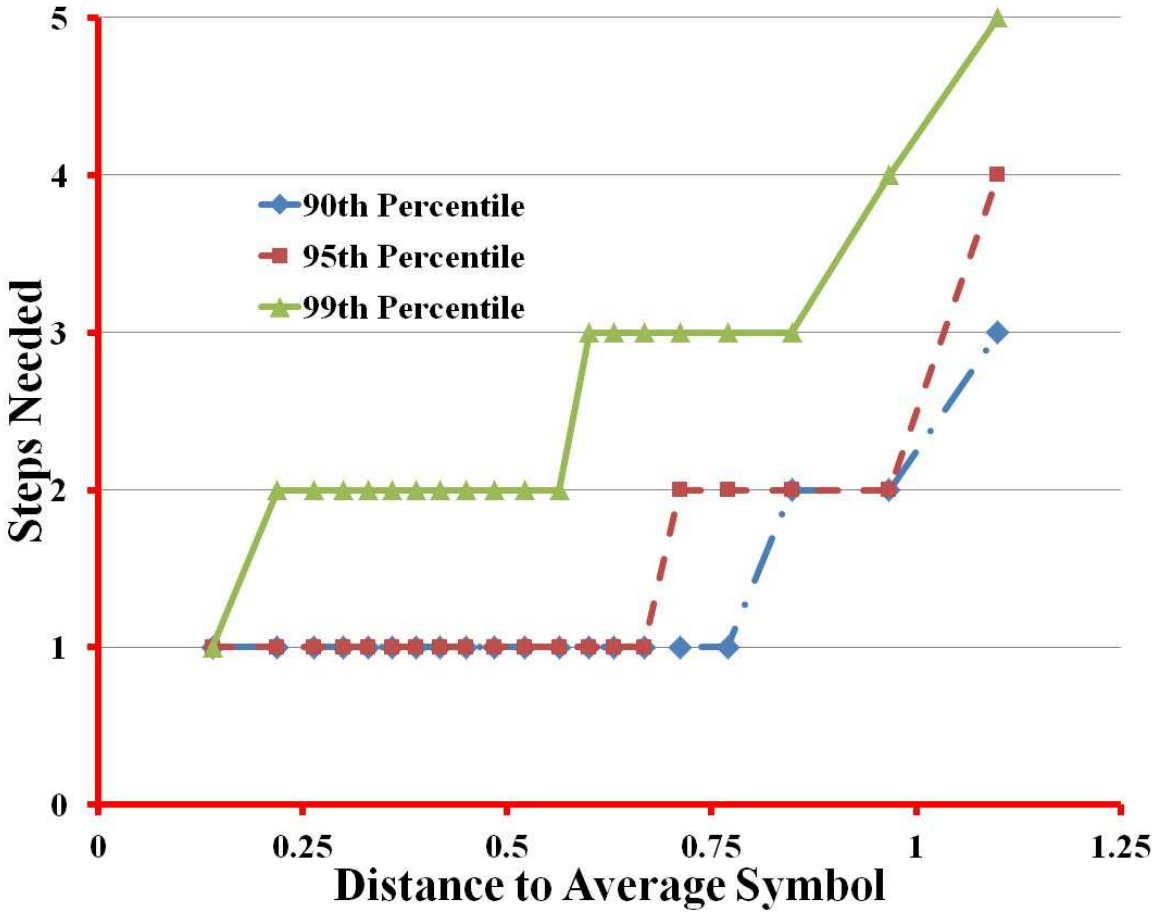


Figure 6.2: Distance to **average symbol** *vs* the number of homotopy steps required. $\Delta s = 0.02$, $\Delta y = 1\%$. The overall success rate was 99.57% (45,440 out of 45,637). Each distance interval in the dense area ($r \leq 0.59$) contains 2800 samples while each interval in the sparse area ($r > 0.59$) contains 1500 samples, except the last one, which contains 1340 samples.

the two points in the function space is given by

$$C(t) = (1-t)C_{ref} + tC_{targ},$$

with t ranging from 0 to 1. The determining points should move smoothly as the character is deformed continuously by the homotopy. By taking discrete steps between the source and the target, we can choose a sequence of intermediary points with each consecutive pair close enough for Algorithm 3 to apply. Figure 5.12 shows an example where Algorithm 3 failed to identify one of the determining points when applied naively. However, when applied in a 5-step homotopy, it succeeds, as shown in Figure 5.13.

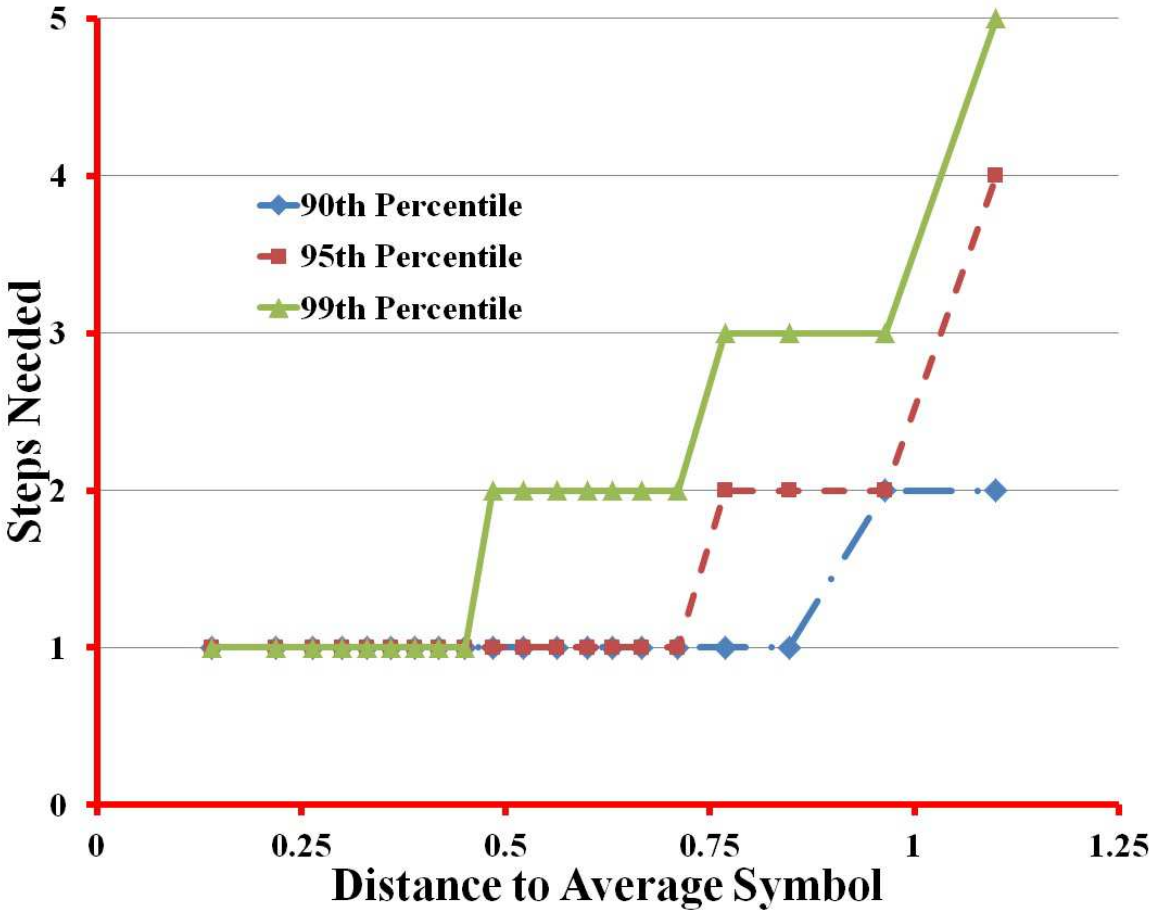


Figure 6.3: Distance to **average symbol** *vs* the number of homotopy steps required. $\Delta s = 0.02$, $\Delta y = 3\%$. The overall success rate was 99.63% (45,470 out of 45,637). Each distance interval in the dense area ($r \leq 0.59$) contains 2800 samples while each interval in the sparse area ($r > 0.59$) contains 1500 samples, except the last one, which contains 1370 samples.

6.4 Experiments

To evaluate the characteristics of the homotopy methods, we have run tests with a moderately large dataset of handwritten mathematical characters. The dataset consisted of altogether 45,637 samples from 240 different symbol types. These samples included Latin and Greek letters, digits, operators, and other mathematical characters, collected from various writers. All of these samples had been classified in advance. As some equivalent symbols were written in different styles, such as completely different forms, different numbers of strokes, or strokes in different orders, the symbols were grouped in a total of 388 classes.

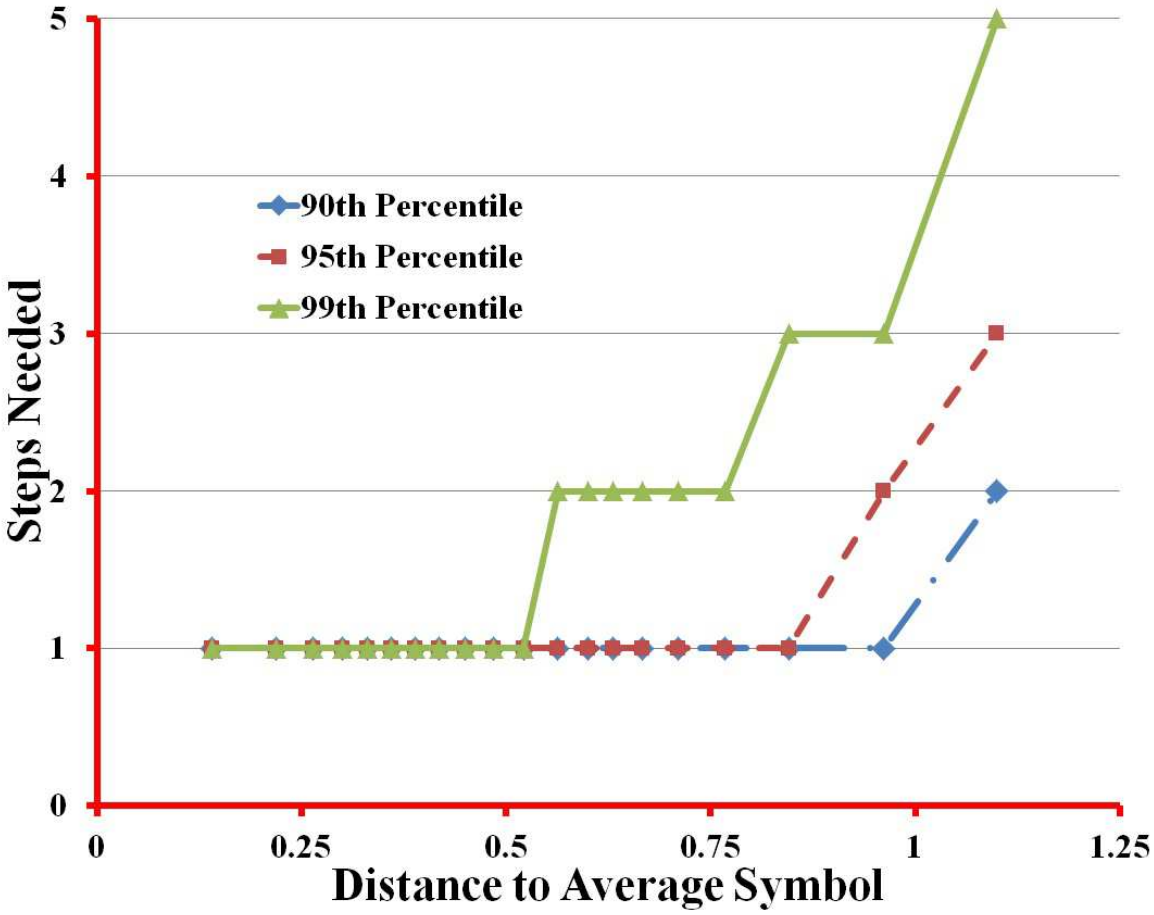


Figure 6.4: Distance to **average symbol** *vs* the number of homotopy steps required. $\Delta s = 0.02$, $\Delta y = 7\%$. The overall success rate is 99.69% (45,496 out of 45,637). Each distance interval in the dense area ($r \leq 0.59$) contains 2800 samples while each interval in the sparse area ($r > 0.59$) contains 1500 samples, except the last one, which contains 1396 samples.

We started the experiments by computing the average symbol of each class, and manually annotating this average symbol with its determining points. We then used this information to try to find the corresponding determining points on all samples in the dataset. We applied a 4-step homotopy strategy as this had achieved a high success rate in earlier work [35] and required a reasonable amount of time to compute. Then we visually inspected the determining points in each of the samples and manually adjusted the few incorrect ones. This collection of samples with corrected determining points then served as the ground truth.

We then went back and tested the data by forgetting the determining point annotations and seeing how many steps it took to recover them. We adopted the average symbol and the nearest neighbour as the reference symbols in the experiments and

evaluated the number of homotopy steps required, respectively. As the locations of determining points were recorded as locations given by the arc length parameter, we compared the values with the ground truth. If the difference was within a particular threshold Δs, we considered it to be correct. Note that this threshold may not work for all the samples as different parameter values may result in the same height of metric lines. For example, given an upper case "T", any point located on the top bar can be used to determine the height of the cap line, but the difference in their parameter values may not fall into the threshold Δs. In such cases, we compared the normalized heights of the metric lines with the ground truth. We considered a result as correct if and only if the difference was within a particular threshold Δy.

We have investigated the relation between the distance to the reference point and the number of steps required under two homotopy strategies. The first strategy performs a homotopy from the average symbol. The second strategy uses a homotopy from the nearest neighbour with known determining points.

Figures 6.2 to 6.4 show, for different thresholds, the number of steps required to correctly identify the determining points using a homotopy from the average symbol. The figures show contours for the number of steps required to obtain certain success rates (90%, 95%, 99%) at varying distances to the reference symbol. Figures 6.5 and 6.6 show the results of similar experiments, but where the reference symbol was the nearest neighbour in the class.

This information may be used to tell how many steps are required to find the determining points in a new sample. After calculating the distance from the new sample to the average symbol, we can read off how many homotopy steps are required for the desired success rate.

Figure 6.2 shows the relation when the tolerances are $\Delta s = 0.02$ and $\Delta y = 1\%$. The overall success rate with no limit on the number of steps was 99.57% (45,440 out of 45,637 samples). We divided the set of samples into groups, based on distance intervals to the average symbol. The size of each interval depended on the density of samples at that distance. There were more samples near the average symbol. In this dense region ($r \leq 0.59$) the intervals were chosen to contain 2800 samples each. In the sparse region, further from the average symbol ($r > 0.59$), each interval contained 1500 samples, except the last one contained 1340 samples. For each interval we determined the 90^{th}, 95^{th}, and 99^{th} percentiles of the number of homotopy steps required to correctly identify the determining points. Given the number of samples in

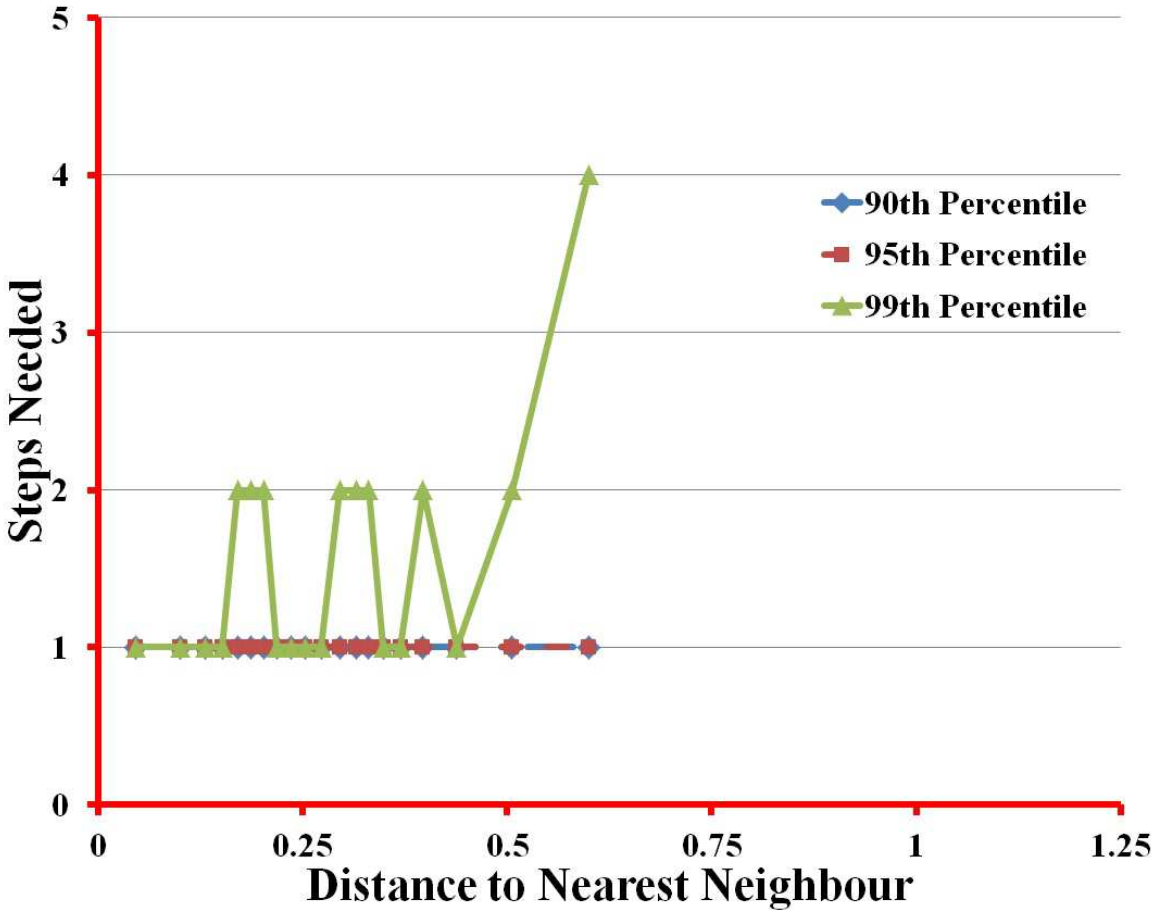


Figure 6.5: Distance to **nearest neighbour** *vs* the number of homotopy steps required. $\Delta s = 0.02$, $\Delta y = 1\%$. The overall success rate is 98.86% (45,116 out of 45,637). Each distance interval in the dense area ($r \leq 0.32$) contains 2800 samples while each interval in the sparse area ($r > 0.32$) contains 1500 samples, except the last one, which contains 1016 samples.

each interval, the margin of error is 2% at a confidence level of 95%. As shown in the graph, as the distance to the average symbol increases, the number of required steps to achieve the 90^{th}, 95^{th}, and 99^{th} percentiles increases. Figures 6.3 and 6.4 show the relation between the distance to the average symbol and the number of homotopy steps needed with larger Δy tolerances.

Figure 6.5 shows the relation between the distance to the nearest neighbour and the number of homotopy steps needed where $\Delta s = 0.02$ and $\Delta y = 1\%$. The overall success rate was 98.86% (45,116 out of 45,637 samples). We divided the distances into a number of intervals, as before. In the dense area ($r \leq 0.32$) the intervals were chosen to contain 2800 samples each. In the sparse area ($r > 0.32$), each interval contained 1500 samples, except the last one, which contained 1016 samples. For each

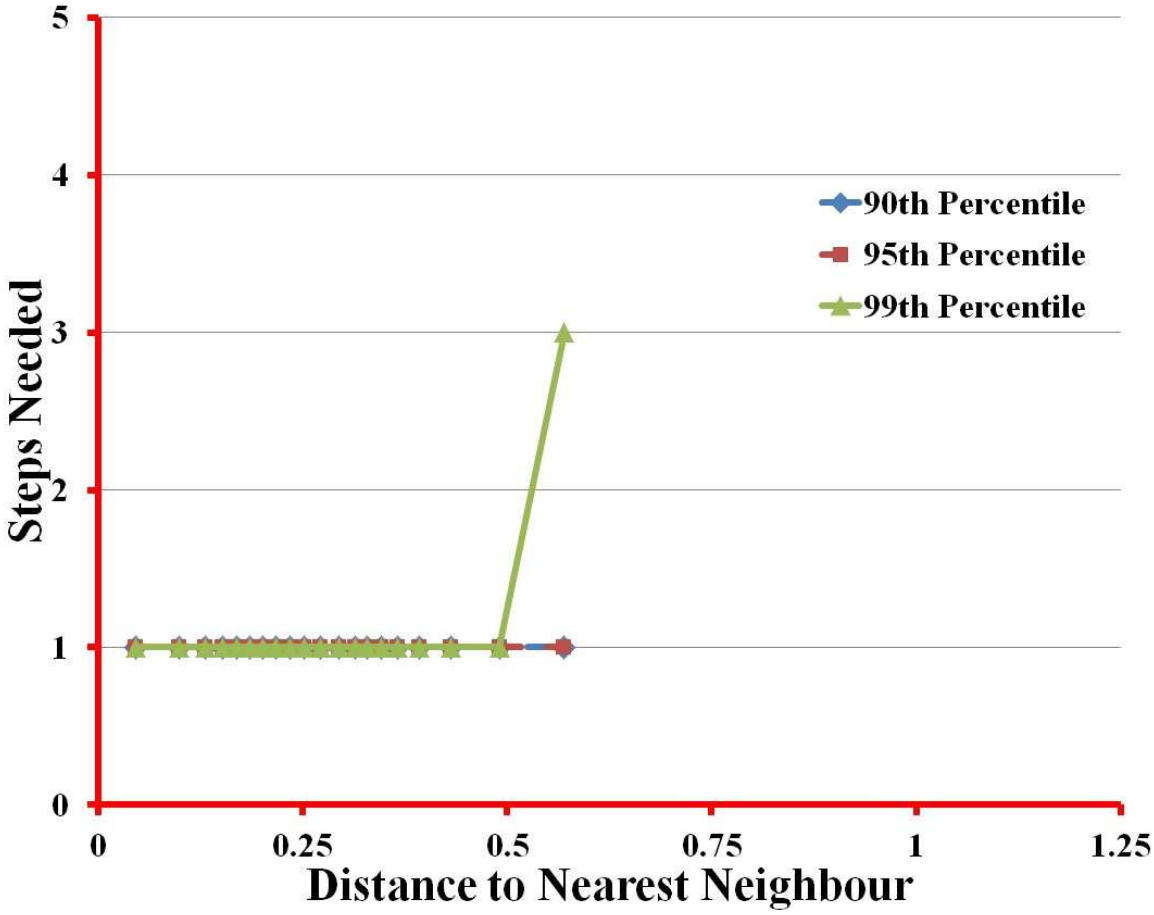


Figure 6.6: Distance to **nearest neighbour** *vs* the number of homotopy steps required. $\Delta s = 0.02$, $\Delta y = 3\%$. The overall success rate is 99.45% (45,384 out of 45,637). Each distance interval in the dense area ($r \leq 0.32$) contains 2800 samples while each interval in the sparse area ($r > 0.32$) contains 1500 samples, except the last one, which contains 1284 samples.

interval we determined the 90^{th}, 95^{th}, and 99^{th} percentiles of the number of homotopy steps required. The margin of error is 2% at confidence level of 95%. As before, the number of required steps generally increased with the distance. Figure 6.6 shows the results when $\Delta s = 0.02$ and $\Delta y = 3\%$. When $\Delta s = 0.02$ and $\Delta y = 7\%$, the graph looks identical to Figure 6.6 except the overall success rate is 99.68% (45,490 out of 45,637). In that experiment, the last interval contained 1390 samples.

We have seen that with both types of reference symbol—average and nearest neighbour—the number of required homotopy steps increases with the distance. Comparing the results, we also see that for a given distance the number of required steps is about the same for both strategies. As is completely expected, the distance to the reference symbol is usually smaller when the nearest neighbour strategy is used.

Therefore with nearest neighbour we have on average a smaller number of required steps. The tradeoff is that we must retain annotations for all symbols in the classifier, rather than just for the average symbol.

6.5 Summary

We have evaluated different strategies to locate special points on handwriting samples by deforming a known instance using linear homotopy on polynomial models.

We have evaluated two strategies for choice of starting point for the homotopy: to use the average symbol of the class and to use the nearest neighbour within the class. We have found that the number of homotopy steps required as a function of distance is about the same with the two strategies. The variance in the number of steps is also similar, but, is perhaps somewhat lower with the nearest neighbour strategy, suggesting a slightly better predictability. At any given tolerance level, the overall success rate was slightly higher when starting with the average symbol than when starting with the nearest neighbour. This does not appear to be significant.

We have also evaluated the number of homotopy steps required as a function of the distance between the new symbol and the starting point in the function space. For the regions where we have sufficient data, we have found a clear relation between the distance in the function space and the required number of steps with both starting point strategies.

In previous work [35], we had used a highly conservative number of homotopy steps (10 to 20) in order to obtain an error rate of 0.25%. The present results allow the choice of an appropriate and much reduced number of steps depending on the homotopy distance (to 1 to 5 steps). In all cases the success rate at correctly locating the determining points on test symbols was greater than 98.8%, so these explorations improve the efficiency of a useful algorithm.

Part III

Collaboration

Chapter 7

Digital Ink Portability

We are interested in the problem of how to support computer-enhanced distance collaboration, where machines can interpret the input and allow useful computations. However, this area has many challenges. The first of these is that existing techniques typically use Application Program Interfaces (APIs) and proprietary ink formats that are restricted to single platforms and consequently lack portability. Collaboration software, however, is most useful when it is platform-independent, both because an individual may use different platforms at different times, and because teams may be composed of members using different systems. Dealing with significantly different client software reduces the ability of the individual or the group to master its use. If any of the members has difficulty, efficiency of the whole team suffers. Therefore, in this chapter we examine the question of how to make both digital ink data and application code portable across multiple platforms.

7.1 Introduction

A number of digital ink applications have been developed in the past years following the emergence of digital ink. Yet most of these applications lack support for portability and are hence restricted to a single platform. Even though cross-platform portability is available, few of them provide many useful ink manipulations on the user interface. We examined the issues of these applications and found the underlying problems to be:

- **The use of platform-specific APIs**
  Applications such as Windows Journal retrieves digital ink by invoking the Windows XP Tablet PC APIs which are restricted to Windows platforms. Such applications can only work on Windows operating systems and are impossible to import to others.

- **The use of proprietary digital ink formats**
  The use of proprietary digital ink formats makes host applications lack portability and limits the ability to interoperate with other applications. For effective data representation and ease of interchange, portable applications should use an ink format standard that is not only neat and elegant but also platform-independent.

- **The complexity of portable capture of digital ink**
  Although pen input devices are able to generate digital ink on almost every individual platform, there has not been a common approach to capture digital ink across all these platforms. To implement a portable digital ink application, developers have to accommodate the detail of each platform. Obviously, this is costly in effort and will eventually affect the design of the user interface.

One of the possible solutions is to construct the application for each platform based on the platform-specific APIs and to interchange the data in a standard format. This, however, will raise other issues. First of all, it increases the time to develop this application, as it requires developers to understand the APIs for each platform, which is very costly in time. Secondly, the application is hard to maintain as it requires maintainers to have a multi-platform background. Last but not least, whenever a new platform comes out, the application has to be reconstructed, from the bottom to the top, based on the new platform's APIs.

7.2 Previous Work

Previous work on digital ink portability has been conducted at the Ontario Research Centre for Computer Algebra. These include the theses of Xiaojie Wu [55] and Amit Regmi [56].

Wu's work focused on the conversions among UNIPEN, JOT and InkML in order

to achieve data portability. Partial platform portability was also achieved in this work. It investigated two ink APIs, the IBM CrossPad API and the Microsoft Tablet PC SDK API, and developed an abstract API on top of them. This abstract API was partially implemented.

Regmi's work was based on the idea of portable multimodal collaboration. It developed a collaborative whiteboard application which provided cross-platform support for Windows, Linux and Mac OS X. It used InkML to exchange data. However, such portability still remained on the data level and was restricted to certain platforms. The implementation of the application varied from platform to platform. The whiteboard client for Windows was implemented in C#. It used the .NET Framework and was thus restricted to Windows platforms. The whiteboard client for Linux and Mac OS X was implemented in Python. Although Python supports cross-platform portability, the client was constructed using Linux-specific or Mac OS X-specific APIs and thus could not be ported to other platforms.

7.3 Objectives

Having recognized all these issues, the primary objectives of our work are to:

- Examine the question of how program code for digital ink applications can be made portable.
- Examine the question of which format is most suitable for digital ink representation.
- Test the suitability of our model of portability using suitably complex cases.

In particular, we wish to have digital ink application code be portable across multiple platforms. Today's platforms capture digital ink in various data structures and report it using different mechanisms. For example, on Windows platforms, digital ink is captured in Wintab packets or `Stroke` objects and reported by the Wintab interface or the Windows XP Tablet PC APIs. The same thing can be done using the `XInput` events and the Linux Input Subsystem on Linux platforms, or the `NSEvent` objects and the Cocoa Framework on Mac OS X. We describe all of these in some detail later. Obviously, portability among such a range of mechanisms requires a

well-structured framework that can handle the details of each platform and model digital ink input in a platform-independent way. Applications that build on top of this framework could then collect digital ink on these platforms without knowing the underlying details. Section 7.4 examines the design issues of this framework.

At the same time, we also wish to have digital ink data be platform-independent. We explain why InkML is most suitable and how it works. Section 7.5 addresses this question of digital ink format.

In summary, we wish to understand what is required to have digital ink applications that we "write once and run anywhere".

7.4 Portability of Software

Today's platforms capture digital ink in various data structures and process it using different mechanisms. Our framework must therefore deal with different lower-level interfaces on different platforms. On Windows, digital ink is captured in Wintab packets or `Stroke` objects and transmitted by the Wintab interface or the Windows XP Tablet PC APIs. A similar scheme is applicable to Linux platforms that use `XInput` events and the Linux Input Subsystem. For Mac OS X we use the `NSEvent` objects and the Cocoa Framework. These platform-specific APIs make development of pen-based collaboration software difficult.

Having explored available APIs on a variety of platforms, including Windows, Windows Mobile, Linux, Mac OS X and Palm OS, we use a framework that can capture digital ink across these platforms and provide a platform-independent, consistent interface to digital ink applications. This framework, as illustrated in Figure 7.1, contains two layers: the platform layer and the Java Native Interface (JNI) layer.

The platform layer receives digital ink input from drivers and passes the data to the upper interfaces: Wintab for Windows, XInput for Linux/Unix and Cocoa for Mac OS. These interfaces push the data to the user space in a platform-specific way and describe each data event inconsistently. This puts the responsibility for event conformance on the shoulders of developers of ink applications. One of our objectives is to allow developers to focus on functionality, rather than the platform-specific issues. This leads to the design of the JNI layer.

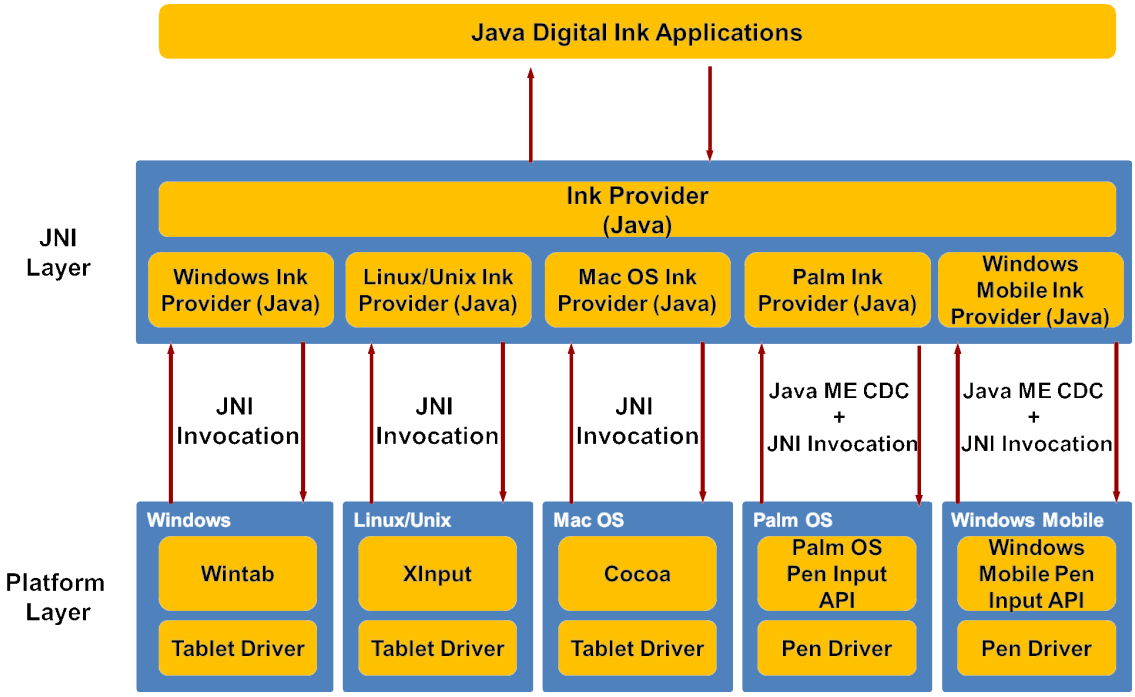


Figure 7.1: A cross-platform framework for digital ink applications

The JNI layer interacts with the platform layer and provides consistent, platform-independent APIs for digital ink applications. As the input APIs are implemented in C-like languages, the JNI layer can invoke them using the Java Native Interface. The JNI layer collects digital ink input from the platform layer, converts it to platform-independent events and then dispatches to digital ink applications. This allows development of pen-based applications on top of the JNI layer to be platform-independent.

7.5 Portability of Digital Ink Data

Another challenge in developing whiteboard-diagramming software is associated with the heterogeneous environment. Various pen devices have different characteristics, including sampling rate, channel properties, screen resolution and so on. The representation of the data generated by a device affects the usability of an application. Previous whiteboard tools did not address this as thoroughly as we require. An example is InkBoard [57], a collaborative sketching application based on Microsoft's Conference XP research platform [58] and designed for Tablet PCs. InkBoard allows design teams to interact by streaming digital ink in Microsoft's proprietary Ink Se-

rialized Format (ISF) [29]. As a result, the application is limited in use to Windows environments where ISF is natively supported.

To represent digital ink, we find it useful to adopt an open, flexible, powerful, platform-, and vendor-independent standard, such as InkML [3]. This allows complete and accurate representation of digital ink by capturing the recording information such as the device characteristics, pen tilt, pen pressure and so on. Most importantly, it provides support for collaboration applications that require streaming ink between participants.

The InkML streaming of digital ink is based on the concept of "context". Whenever digital ink is written, there is some context in effect. Contexts may be represented externally using the `<context>` element in InkML. This can contain various associated attributes, including canvas properties, canvas transformation, trace format, ink source metadata, and time stamp. Initially, each ink collaboration participant obtains a default context and listens for context changes. This is similar to an event-driven model in which context changes are made when contextual elements are received. In practice, these elements will intersperse among digital ink streams, as shown in Figure 4.2.

With this model, each participant can easily maintain the current context of the sender in addition to its own local context. Whenever a new contextual element is received, it simply updates old values. Contextual elements are sent only when there is a context change, which helps to decrease streaming overhead on the wire. The overhead can be further reduced by making references to existing contextual elements, which can be pre-defined or previously received. This can be accomplished by using referencing attributes (e.g. `brushRef`) of `<context>`.

7.6 Summary

The primary objective of this chapter was to explore the dimension of digital ink portability, both of digital ink data and of digital ink application code. Our goal was to be able to have applications that we can "write once and run anywhere".

To support cross-platform viability, we have presented a framework that can handle the details of a variety of platforms and provide a consistent, platform-independent

interface for digital ink applications. Applications that are built on top of this framework can collect digital ink across all the supported platforms without knowing their details.

To support platform-independent data representation, we have evaluated several ink format standards including JOT, Unipen, ISF, SVG and InkML. We have chosen InkML as our data representation as it provides both digital ink streaming and archival support independent of platforms.

There are a number of interesting directions that can be pursued in the future. We plan to explore digital ink portability on platforms without Java such as the iPhone OS. We are also interested in issues that would arise in introducing video and text support.

Chapter 8

Multimodal Collaboration

We investigate the question of how multimodality can be used in computer-mediated mathematical collaboration. The primary objective is to analyze the design issues in incorporating multimodal interactions in this kind of collaboration. This chapter is based on the article "InkChat: A Collaboration Tool for Mathematics" [28] co-authored with Stephen M. Watt, that appeared in the electronic proceedings of 2013 Workshop on Mathematical User Interfaces.

8.1 Introduction

A number of computer applications have been developed over the past years to accommodate the needs of collaboration. One general category of these is whiteboard systems, where multiple users can interact over a shared canvas using a particular input method. For example, the use of pen input allows participants to write or draw naturally, which has great potential to increase productivity, especially in mathematics [59]. Existing systems, however, typically allow only one input method to be used at a time. For example, in Microsoft OneNote, one can either type or draw, but not simultaneously. This places strong limitations on what can be done. For example, it becomes quite awkward to explain an activity while it is being performed.

Mathematical collaboration software can be most useful when it supports input from multiple modalities, as it allows collaborators to interact more richly and participants to use the input methods that are most suitable. For example, in mathematics,

some points can be most efficiently made through the spoken word while others can best be communicated by a hand-drawn diagram or equation. Software for collaboration also should allow users to perform various editing operations to revise or tidy up jointly created drawings, text and so on. Finally, collaborative work usually includes exploring ideas in discussions that can have false starts and dead ends so the ability to record and roll-back the discussion to prior points is important.

8.2 Previous Work

Considerable related work has been conducted, some of which we highlight here. In the 1990s, QuickSet [60] was a multimodal framework used by the US Navy and US Marine Corps to set up training scenarios and to control virtual environments. It accepted voice and pen input, communicating via a wireless LAN through an agent architecture to a number of systems. The system could recognize voice input with certain responses. If voice interaction was not feasible, it could still analyze digital ink and then give several possible interpretations. This demonstrated that multimodal interactions could enable efficient communication.

LiveBoard [61] was developed at the Xerox Palo Alto Research Center in 1990 to support slide presentations and group meetings. It provided a large display system with the ability to interact with a multi-state cordless pen. LiveBoard used a paint application to render digital ink generated by the pen, and then projected onto the large display. It was able to record the drawings and recover them on any LiveBoard. On the other hand, it was not possible to perform collaborative operations at a stroke level.

Tivoli [62] was an extension of LiveBoard. It overcame the limitations of LiveBoard using stroke objects, as opposed to pixel map images. These stroke objects were represented and manipulated by a proprietary Software Development Kit(SDK). In order to improve the collaboration productivity, Tivoli supported multiple cordless pens for multiple users.

Classroom 2000 was an application [63] whose primary purpose was to create an environment to capture as many activities as possible from the classroom experience. It included tools to automate the production of lecture notes and to assist students in

reliving lectures. This application did not support real-time distributed collaboration, however.

Inkwell [64] was a handwriting recognition application developed by Apple Inc, and restricted to Mac OS. Inkwell was able to accept handwriting input and then convert it into recognized text in English, French and German. Inkwell could also be used to draw a sketch and insert it to any place that allowed a picture. Nonetheless, Inkwell used platform-dependent APIs and therefore could not be used on other operating systems.

InkBoard [57] was a network-shared whiteboard application which was released in 2004. It allowed graphical collaboration and designs, including network-shared ink strokes and audio/video conferencing. As it integrated the Microsoft Conference XP technology, it was restricted to Windows platforms.

In the same year, Electronic Chalkboard [65] was developed. Its goal was to integrate distance education tools with the traditional blackboard experience. It could load images and interactive programs from a file system or the Internet and could interact with computer algebra systems and display computation results. Because content was saved as images, it was not possible to later edit or perform semantic operations on saved sessions.

Windows Journal was a note-taking application developed by Microsoft. It had been integrated in Windows XP, Windows Vista and Windows 7. The user interface supported many operations, such as switching among pens, highlighters and erasers, and moving items around the page, etc. As Windows Journal used the Windows XP Tablet PC SDK [66], it was restricted to Windows platforms and thus could not be used on other operating systems. Windows OneNote is another Windows application still in use. It is similar to Windows Journal, but is more elaborate and suffers.

Inkscape [67] was an open source vector graphics editor application. It was able to accept digital ink from a pen and save it using Scalable Vector Graphics (SVG) [68] format. SVG is an XML-based data format for describing two dimensional vector images. Compared to bitmap images, SVG images have many advantages such as smaller file size, resolution independence and ease of resizing. SVG is a very general data format for all kinds of geometric objects, but it does not provide direct support for the needs of digital ink.

To explore ideas for mathematical collaboration, Regmi and Watt [69] developed

a whiteboard application that provided collaborative sessions with synchronized voice and digital ink on a shared canvas. This system could save sessions with the digital ink in InkML format and the voice as an MP3 sound file. A significant drawback, however, was that the client interface implementation varied from platform to platform: The client for Windows was implemented in C#, using the .NET framework, while the client for Linux and Mac OS X was implemented in Python. Although Python supports cross-platform portability, the client was constructed using Linux-specific and Mac OS X-specific APIs and thus could not be ported to other platforms.

To address the portability issue, Hu, Mazalov, and Watt [70] proposed a streaming digital ink framework for multi-party collaboration. The framework was portable across multiple platforms and consisted of a number of extensions, These extensions could work independently and simultaneously, serving as plug-ins for the host collaboration software. It is portable across multiple platforms including Windows, Linux, and Mac OS X. It currently uses the popular Skype and Google Talk services as the backbone to deliver data streams, but other transport mechanisms could be used. The digital ink data is represented in InkML, allowing flexible manipulations for different content types, such as mathematics and diagrams. The collaborative sessions can be recorded and stored for playback, analysis or annotation.

8.3 Multimodal Collaboration

Multimodal input is useful as it provides versatile means for users to interact with computers. These input modalities include keyboard, mouse, voice, pen, and video and so on. We will focus on the voice input and pen input in this section, and explore their capabilities in mathematical collaboration.

8.3.1 Voice Input

Voice communication is a fast and natural way to interact with other people and computers. Most people speak faster than they can type or manipulate a mouse. Notably, certain people with physical disabilities prefer operating their computers simply by speaking. Voice input is hands-free, which is useful if one is driving. Also, voice input is flexible. One does not have to sit in front of a computer: it is possible

to use voice input while active or while sitting, standing, or reclining. Most modern devices come with built-in microphones.

8.3.2 Handwriting Input

With the widespread availability of pen-based devices such as Tablet PCs, PDAs and even cell phones, pen input starts playing an important role in human computer interaction. Pen input is a natural and powerful input modality since everyone learns to write in school. It is versatile as it provides more gestures and motions available compared to mouse and keyboard input. Many devices without pens support touch input, which may also be used to capture handwriting using a finger, but at a lower resolution.

Handwritten input is expressive. Modern devices capture digital ink traces in a two dimensional writing plane, and may support pressures angles and non-contact pen height. This may be used to capture mathematics, as most mathematical notations are two dimensional, with similarities to both text and drawing. Mathematical formulae are hard to understand by means of voice, keyboard or mouse, but can be expressed easily in handwriting.

8.3.3 Voice and Handwriting Multimodal Input

We chose voice and pen as the input modalities for collaboration. Together they provide more than either individually, and indeed we find in this combination the whole is more than the sum of its parts. The advantages of voice and pen multimodal collaboration include:

Portability Most computing platforms support both voice and pen input.

Ease of use Writing and voice are familiar and require little training to use.

Complementarity Both input modalities can work independently and simultaneously. Speaking and writing at the same time gives two communication channels, allowing one to be used to explain or amplify the other. Either may give the main message, with the other supporting it, or both be equally important. Two channel

communication avoids the clutter and confusion that arises when two related messages are multiplexed through one modality. For example, text with footnotes or parenthetical remarks or formulae with many annotations.

8.4 Summary

We have analyzed the design issues in incorporating multimodal interactions in mathematical collaboration. We have chosen voice and pen as our input modalities in collaboration as they together add more benefits than either alone. We have also adopted InkML to represent digital ink as it allows real-time streaming in heterogeneous environments. To demonstrate our ideas, Chapter 10 will present InkChat, a whiteboard application, which can be used to conduct collaborative sessions on a shared canvas. It allows participants to use voice and digital ink independently and simultaneously, which has been found useful in mathematical collaboration.

Chapter 9

A Streaming Digital Ink Framework for Multi-Party Collaboration

We present a framework for pen-based, multi-user, online collaboration in mathematical domains. This environment provides participants, who may be in the same room or across the planet, with a shared whiteboard and voice channel. The digital ink stream is transmitted as InkML, allowing special recognizers for different content types, such as mathematics and diagrams. Sessions may be recorded and stored for later playback, analysis or annotation. The framework is currently structured to use the popular Skype and Google Talk services for the communications channel, but other transport mechanisms could be used. The goal of the work is to support computer-enhanced distance collaboration, where domain-specific recognizers handle different kinds of digital ink input and editing. The first of these recognizers is for mathematics, which allows converting math input into machine-understandable format. This supports multi-party collaboration, with sessions recorded in rich formats that allow semantic analysis and manipulation of the content. This chapter is based on the article "A Streaming Digital Ink Framework for Multi-Party Collaboration" [70] co-authored with Vadim Mazalov and Stephen M. Watt, that appeared in the proceedings of the 2012 Conferences on Intelligent Computer Mathematics.

9.1 Introduction

We are interested in development of an infrastructure that helps researchers, engineers, teachers and students to collaborate online over pen-based and graphical interfaces. Pen-based collaboration in mathematical domains can significantly increase productivity, e.g. Gowers, et al. conducted an experiment of "attacking" a mathematical problem through a collaboration of volunteers online [71]. In just over five weeks, more than two dozen individuals contributed approximately 800 comments that led to the solution of the problem. We believe that the synergy of pen-based *collaboration* and *recognition* of mathematical input can enhance the efficiency of online interaction. Nevertheless, there is no technology that allows to capitalize on both simultaneously: some software handles recognition without the ability for real-time sharing, e.g. the Maple computer algebra system [13], while other systems provide a whiteboard for collaboration, but no mathematical recognition, e.g. Microsoft OneNote [14], Calliflower [15] or Dabbleboard [16].

9.2 Objectives

We present a framework for multi-user online collaboration in a pen-based and graphical environment. This environment allows participants conducting and archiving collaborative sessions that involve synchronized voice and digital ink on a shared canvas. The digital ink is represented as InkML, allowing special recognizers for different content types, such as mathematics and diagrams. The collaborative sessions may be recorded and stored for later playback, analysis or annotation. The framework currently employs the popular Skype [72] or Google Talk [73] services as the backbone to deliver data streams, but other transport mechanisms could be used. Our objective is to support computer-enhanced distance collaboration, where domain-specific recognizers handle different kinds of digital ink input and editing. The first of these domains is mathematical input, which has similarities to both natural language handwriting input and two-dimensional diagramming. This framework has potential to increase productivity in teleconferences or to enhance online learning and tutoring, as the participants can interact in a natural way. A version of the framework implementation is available for download at `http://www.orcca.on.ca/InkChat/`. The current release (version 0.9.5) has many of the features described in the paper. Some

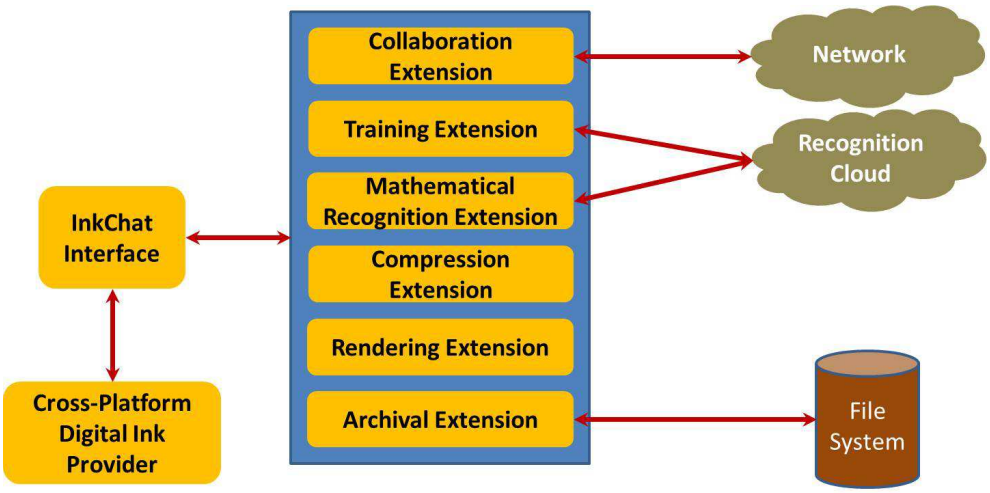


Figure 9.1: InkChat architecture

of the features described here have been implemented in separate packages and will be integrated in later versions of InkChat. This work is an outgrowth of earlier work [74] that allowed collaborative inking, but which did not allow modular extension.

The remainder of this chapter is organized as follows. Section 9.3 presents a high-level overview of the architecture of the framework and gives details on each component. In Section 9.4, we describe the implementation of the framework. A case study and a discussion of lessons learnt are given in Section 9.5. Section 9.6 summarizes the chapter.

9.3 Architecture

We now present the framework for multi-user online collaboration, allowing sessions to be recorded in formats that allow semantic analysis and manipulation of the content. The framework currently consists of InkChat, a digital ink application developed at Ontario Research Centre for Computer Algebra, and a number of extensions. InkChat is the main platform of the application, on top of which other extensions may be added. Figure 9.1 presents a high-level overview of the architecture of InkChat. InkChat interacts with the cross-platform framework, presented in Section 7.4, whose primary purpose is to collect digital ink from a variety of platforms and to provide a platform-independent, consistent interface for digital ink applications. The six

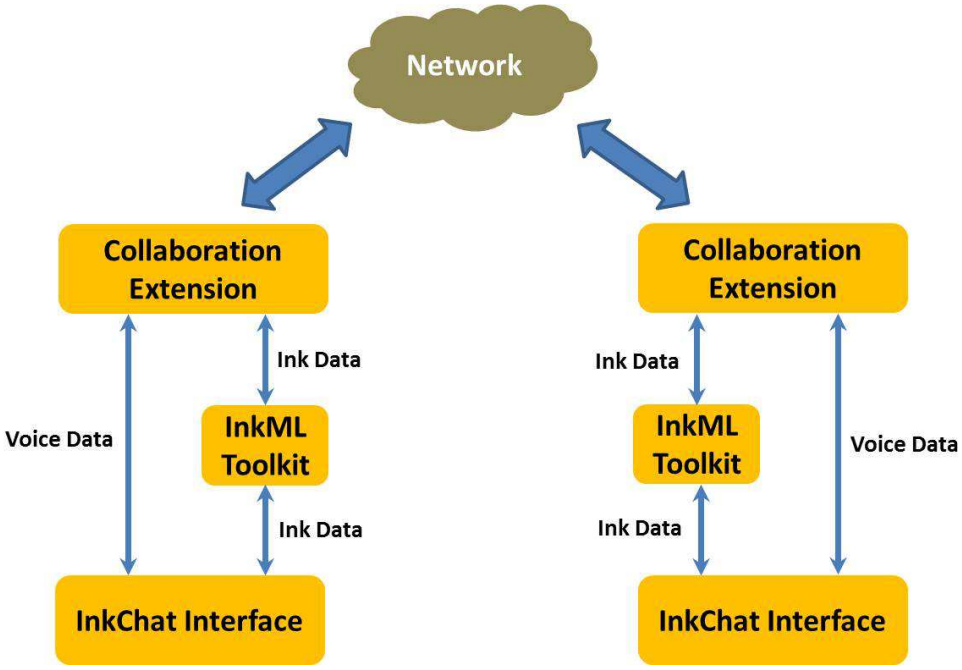


Figure 9.2: Collaboration framework

extensions, most of which can work independently and simultaneously, serve as plug-ins for InkChat. We outline the details of each extension below.

9.3.1 The Collaboration Extension

Similar to the traditional concept of sharing a session between several parties available in communication software (e.g. having a group call or a text chat), the collaboration extension allows real-time sharing of the canvas among participants and enables the participants to edit content on the canvas. Synchronization of the canvas is performed through the underlying communication backbone. In addition, the voice and video channels are typically available through the communication software as well. The collaboration scheme between two clients is shown in Figure 9.2. Digital ink data is first converted into InkML format and then sent in a data channel parallel to the voice channel. The collaboration extension itself is an abstract layer that can handle different network protocols including pipes and sockets in P2P (peer-to-peer) and the traditional client-server configurations. To adapt to the collaboration environment, we allow participants to choose the most suitable protocol at the beginning of the conversation. For example, if the collaboration is intended to take place in a small group and requires only the basic functions of whiteboard, P2P would be preferable

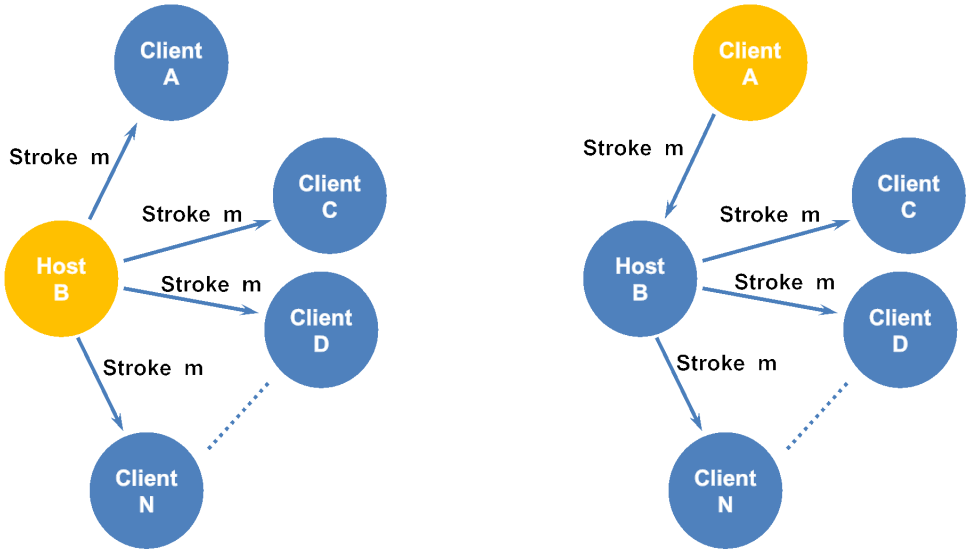


Figure 9.3: Sessions with the stroke initiated by (a) the host and (b) the client.

as it is inexpensive and fairly simple to set up and manage. If the collaboration is intended to comprise a large number of participants and demand sophisticated computing services (e.g. mathematical formula simplification), client-server mode may be more appropriate.

InkChat supports conference mode where more than two participants can be involved in one conversation. Depending on the chosen network protocol, InkChat employs different mechanism to communicate with other participants. When a P2P backbone such as Skype is used, the conference is initiated by the host that has a connection with every other participant. Digital ink routing shares the same mechanism as audio routing, each ink stroke will be broadcast by the host to all participants except the initiator. Figure 9.3 shows the two cases, when a host and when a client initiate a stroke. In the client-server network, the server will play the role of the host and establish a connection with each client, as shown in Figure 9.4. Both the digital ink and audio data are sent to the server first and then broadcast to all the other clients.

9.3.2 The Training Extension

In an adaptive recognition environment, a separate training phase is not required. Having some number of training samples in each class can, however, significantly

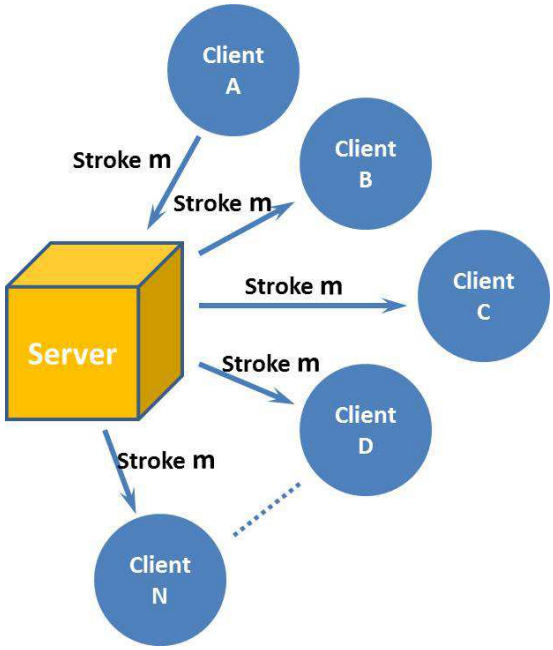


Figure 9.4: A client-server configuration with a stroke initiated by a client.

improve the initial recognition. The number of training samples that a class should contain depends on the recognition methods. Using our approach, the recognition rate depends on the minimal distance to convex hulls of neighbouring classes. For most of the classes in our dataset, about 20 classes are required [75].

A training session may be initiated on first use of the application or when a new character is first introduced. Once training is finished, a profile of classes and training samples is saved as an XML document. The profile is a hierarchical container of catalogs (groups of related characters, e.g. digits, Latin letters), symbols, writing styles, and training samples [75]. Training samples are divided by different forms for symbols (e.g. a one-stroke versus a two-stroke numeral "ϕ"), since many recognition methods are sensitive to the direction and order of strokes. The training extension also provides an interface for the user to manage profiles of training samples.

9.3.3 The Mathematical Recognition Extension

The mathematical recognition extension is an ongoing project. The objective is to have domain-specific recognizers that handle different kinds of digital ink input. We currently focus on the classification of handwritten mathematical symbols, as the essential component of math formula recognition, and spatial analysis of characters.

In our classification paradigm, each character is represented as a single point in a space of curves. The coordinates of this point are the coefficients of truncated orthogonal series approximating the coordinate functions of the trace [10]. Classification of a character is based on the distances to the convex hulls of nearest neighbours in this space. A sequence of incoming strokes is divided to form characters with the highest classification confidence. Spatial analysis of samples is performed based on the relative positioning of the centers of mass of adjacent characters.

This approach typically does not require many training samples to discriminate a class. However, because there is a large number of classes, the training dataset may contain tens of thousands of characters. The dataset evolves with each recognized sample. Synchronization of the evolving per-user training exemplars across several pen-based devices may become tiresome. To address this aspect, the storage of the training database can be delegated to a cloud [75]. At present, recognition is always done locally, although this could alternatively be done by a server. Locally, the recognition extension accepts raw ink from InkChat and pre-processes it. This preprocessing typically includes computing approximation of the character and normalization with respect to the size and position. Consecutive strokes are merged into candidate symbols and approximated, yielding points in the space of curves. The coefficients are recognized [10] by the recognition extension. A ranked list of recognition results is sent to InkChat for display and confirmation or rejection/selection. The result is stored remotely as classification experience for subsequent adjustment of training clusters.

9.3.4 The Compression Extension

The compression extension implements a hybrid of functional [76] and linear [77] approximation. There are several schemes for segmentation of digital ink for compression by the functional approximation method. The most robust is the adaptive segmentation that dynamically selects the degree of approximation and the size of coefficients. Coefficients are recorded as floating-point numbers with base 2 [76]. The functional approximation is best-suited for curly handwriting, for example as with math symbols.

Linear approximation allows compact representation of nearly linear segments, and is therefore suitable for many forms of representation of mathematical and engi-

neering knowledge, such as graphs, tables, or diagrams. The method removes points that least affect the shape of the curve, so long as the error between the original and the approximating curves remains within a threshold. The method can be viewed as a dynamic adjustment of the density of points, depending on the shape of the stroke. The method is very fast and yields reasonable compression.

9.3.5 The Rendering Extension

Different rendering styles are required for different drawing and writing activities. For example digital painting and Chinese calligraphy may require an instrument that behaves like a paint brush, while diagramming may best be done with an instrument that behaves like a pen or pencil. To support these needs, different brush models may be selected.

Round Brush The round brush, as its name suggests, draws each ink point as a filled circle. We use three parameters to model the round brush: x, y to indicate the position and r to measure the circle radius which can be a function of the pen tip pressure. We fill the gaps between the circles of consecutive points by forming envelopes with circle tangents, as shown in Figure 3.4.

Tear Drop Brush The idea of the tear drop brush is to model the contact of a brush head as it varies in distance to the canvas and is dragged along. This contact area is modelled as tear drop – a circle and the area enclosed by the tangents to a trailing point (which may degenerately be on the circumference). Varying the radius models what happens when the brush varies in distance to the canvas, and the trailing point is determined by the past history of the brush. There are five parameters, as shown in Figure 3.5: x and y give the position of the ink point, r gives the head radius, θ the direction of the tail, and ℓ the length of the tail from (x, y). The tear drop brush model has been adopted by InkML [3] and the calculation of r, θ and ℓ from the stream of x, y and pressure values has been discussed in [24, 78]. As with the round brush, strokes are rendered by filling brush shapes and the area formed by tangents between successive brush outlines.

9.3.6 The Archival Extension

InkChat stores handwritten input using InkML format. Strokes are converted to an InkML stream as they are captured. They are then saved to the current ink session before being sent to other participants. As the stream is received by a participant, InkChat immediately parses the stream and saves the strokes to the current ink session. When the conversation is finished or the user requests to save, InkChat writes the current ink session to an InkML file with the ink from all participants. InkChat also supports loading collaboration sessions from InkML documents.

9.4 Implementation

We have used the framework presented in section 9.3 to build the InkChat application. Most components are implemented in Java as it provides strong portability and applications can run on any Java Virtual Machine regardless of system architecture. For platforms that do not support Java directly, the Java code can be compiled to C. Our goal to maintain a single, coherent source that can be compiled for all platforms. Below we briefly describe the implementation of the major components.

9.4.1 User Interface

The InkChat user interface is illustrated in Figure 9.5. It is designed to have buttons grouped so that the distance of moving the pen is minimized. Users can write, erase, and highlight by selecting corresponding brushes. Editing is also supported. Users can redo, undo, cut, copy, and paste different kinds of content, such as images, typed-text, and digital ink. In order to better support collaboration, we have provided a floating pointer and a page navigator. The floating pointer can be used to point at target objects on the shared canvas without leaving ink. Together with voice channel, this allows pointing to and discussing aspects of the common canvas. The page navigator allows participants to create new pages or review earlier pages. Once a session is finished, all pages can be recorded and stored for later playback.

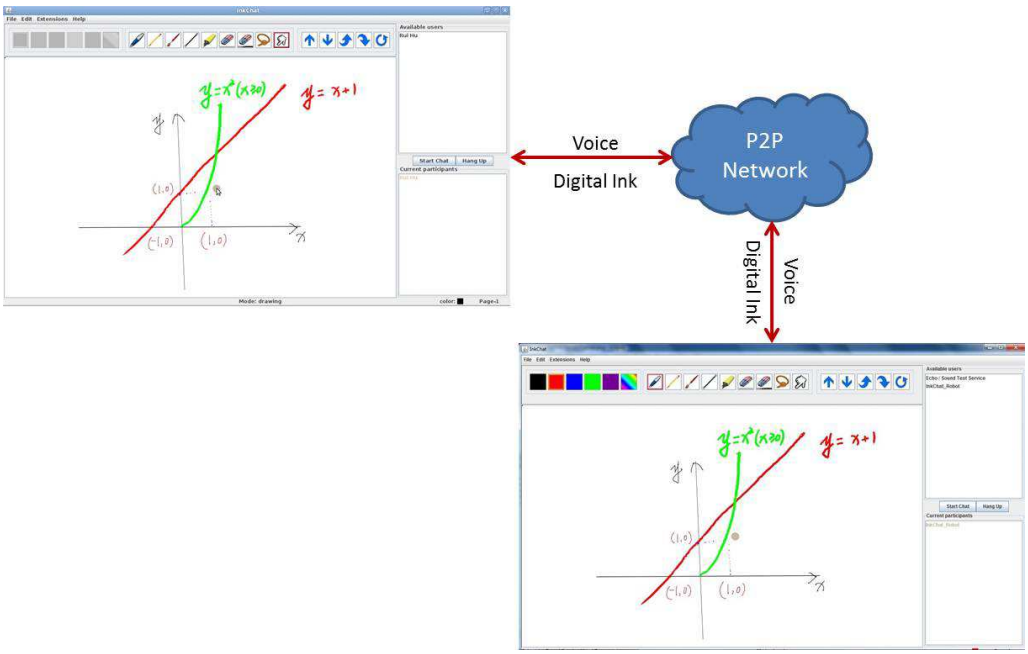


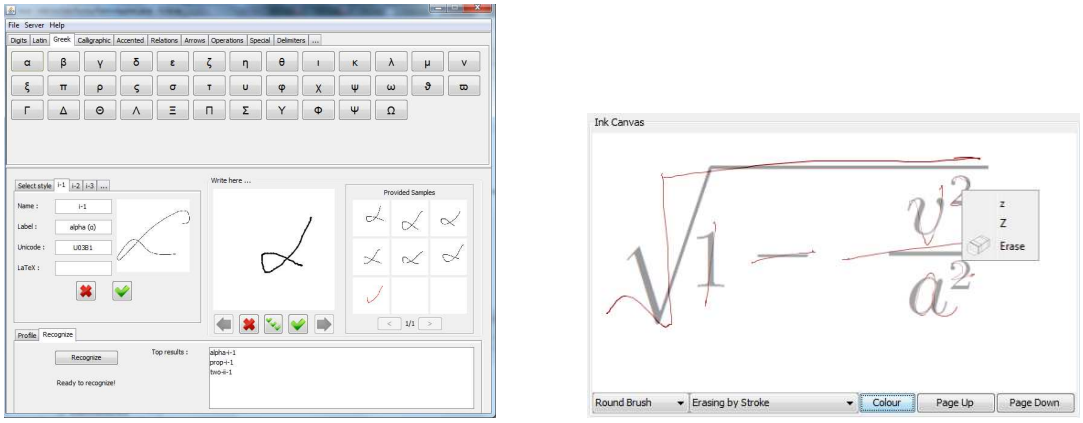
Figure 9.5: An example of diagramming in InkChat with Skype service.

9.4.2 Collaboration

The InkChat software combines digital ink and voice in a multi-user conversation with a shared canvas. It is structured so it can use the popular Skype and Google Talk services for transmitting data streams. InkChat can be used by people working together in the same room or on opposite sides of the planet. Sessions may be archived to be resumed later or for later processing. An example of collaboration using InkChat is illustrated in Figure 9.5. Two participants, the author and user `InkChat_Robot`, are involved in the collaboration using Skype. The author is working on a Windows machine while the `InkChat_Robot` is on a Linux machine. The top left screenshot shows the `InkChat_Robot`'s canvas and the bottom right screenshot shows the author's canvas. Remarkably, the `InkChat_Robot` is using the floating pointer to point around coordinate $(1, 1)$. This is streamed to the author's canvas in real time.

9.4.3 Training

The training extension presents an extensible catalog of symbols and styles, organized in a tabbed panel, as shown in Figure 9.6(a). Each tab contains a list of symbols. Once the user selects a symbol, the panel with styles becomes available. Styles are shown



(a) (b)

Figure 9.6: The recognition interface: (a) the training extension and (b) the mathematics recognition extension.

as animated images for visualization of stroke order and direction. Each sample is associated to an existing or new style. If a style has not been selected, it is determined automatically based on its shape and the number of strokes. All the elements of the interface (catalogs, symbols, styles and samples) are dynamic: A context menu is allows the user to create, delete or merge elements. A profile can be saved locally or synchronized with a server.

9.4.4 Mathematical Recognition

Classification of symbols takes place when a user performs handwritten input through InkChat and the recognition extension is enabled. For each character, a context menu is available that lists the top recognition candidates, see Figure 9.6(b). If the user chooses another class from among the candidates listed in the context menu, adjacent characters can be reclassified based on the new context information. There are two usual approaches to expression recognition: character-at-a-time (each character is recognized as it is written) and expression-at-a-time (characters are recognized in a sequence, taking advantage of the context) Our current approach lies between these: we are experimenting with recognition within a moving window of context and consider all ink outside that window to be “dry” and recognized. Classification results can be displayed super-imposed on the digital ink or can replace it.

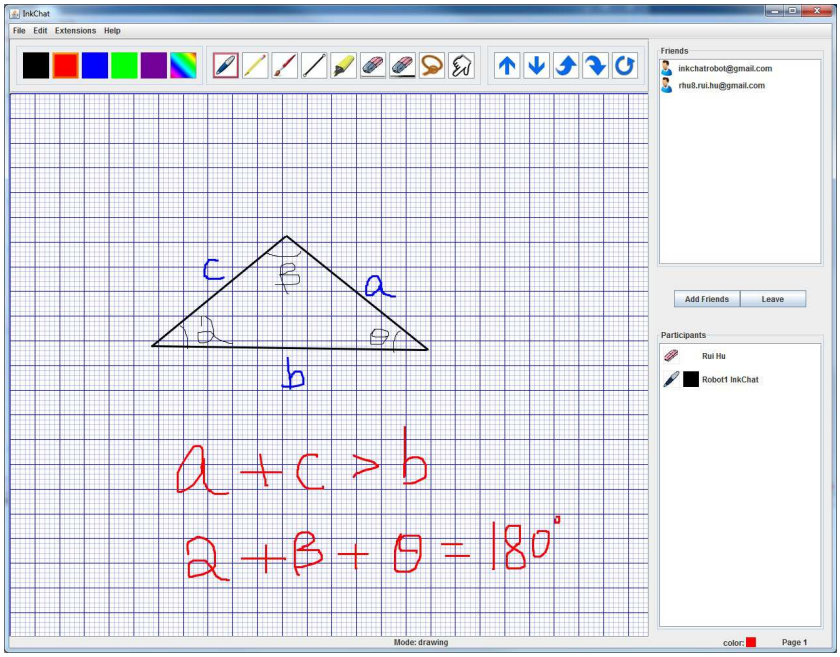


Figure 9.7: An example of online learning and tutoring

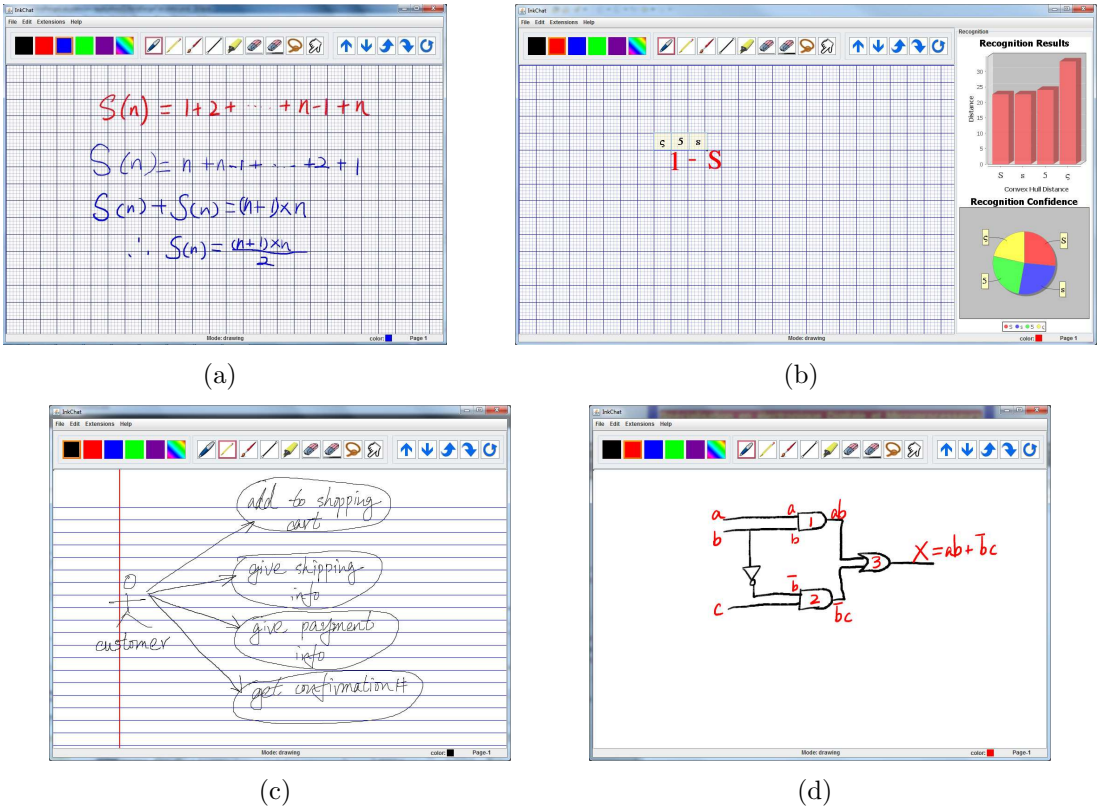
9.5 Discussion

9.5.1 Scenarios

Here we describe several cases in which the framework has been found useful.

Online learning and tutoring Figure 9.7 shows the triangle inequality being discussed by three persons. The top-right panel lists the available friends of the user and the lower-right panel shows the participants who are currently in the session (excluding the user). The Google Talk service is employed in this session. The triangle was drawn by the participant `Robot1 InkChat`. It was later annotated by the user to better explain the triangle inequality.

Collaborative work on math and diagrams The framework provides a collaborative environment where participants can interact through a shared canvas and a voice channel. Figure 9.8 shows examples in different domains. The ink and meta-information is shared between participants through Skype or Google Talk.



(a) (b)

(c) (d)

Figure 9.8: Examples of math and diagram collaboration (a) formula induction, (b) math symbol recognition, (c) use case diagram, (d) circuit diagram.

Demonstration of animated diagrams At least in some cases, animation can significantly improve understanding of certain complex diagrams [79]. The InkChat framework allows digital ink to be animated, reproducing the original writing sequence, or sequenced in some other manner. Thus, animated sketched diagrams can be created and shared in real-time or made available for download.

9.5.2 Lessons Learnt

The architecture we have presented is a multi-year ongoing project, and there are several important lessons that we have learnt

- **Lesson 1:** *Anticipate that the world will keep changing.* It is important to have the data streams for an application well specified so new technologies, such as JavaScript and HTML 5 canvas can participate.
- **Lesson 2:** *There is power in less capable, but more uniform, interfaces.* We

initially had different interfaces for ink sharing, recognition and the other behaviours. Having a common interface, while not perfectly adapted to any of these tasks, allowed them to be combined flexibly in useful and unanticipated ways (e.g., combining the recording/playback and recognition modules).

- **Lesson 3:** *Plan on sharing.* The extensions and the main platform should be designed to allow sharing of the functionality with third parties through the mechanism of web services. For example, the recognition extension should be able to accept a SOAP message with coefficients of a character to be classified and send back the recognition candidates.

- **Lesson 4:** *Don't pin the user down.* Plan on individuals making use of many devices. This implies a protocol for network storage of user-specific information, such as personalized handwriting recognition data.

- **Lesson 5:** *Anticipate standards.* Tracking draft standards prior to their maturity, and indeed participating in the standardization process, can be an effective strategy to achieve interoperability. The project was using InkML before it was recommended as a standard by W3C. Today InkML allows cut-and-paste of digital ink between applications, e.g. between Microsoft Office 2010 and InkChat.

9.6 Summary

We have presented a framework for pen-based, multi-user, online collaboration applicable to mathematical domains. The framework supports teamwork on scientific and engineering problems by allowing participants to make contributions to a shared canvas while having a discussion. The framework is portable and uses InkML, a W3C standard, as a representation format. The architecture of the framework is extension-based with InkChat as the main component. On top of InkChat, several extensions have been developed for collaboration, training, mathematical recognition, compression, rendering and archival. We have presented a high-level overview of these components. In ongoing work, we are particularly interested in improved recognition of mathematical formula and architecting more powerful combinations of extensions.

Chapter 10

InkChat: A Collaboration Tool for Mathematics

To demonstrate our ideas of digital ink collaboration, we present InkChat, a whiteboard application, which can be used to conduct collaborative sessions on a shared canvas. It allows participants to use voice and digital ink independently and simultaneously, which has been found useful in mathematical collaboration. This chapter is based on the article "InkChat: A Collaboration Tool for Mathematics" [28] coauthored with Stephen M. Watt, that appeared in the electronic proceedings of 2013 Workshop on Mathematical User Interface.

10.1 InkChat

We have developed a platform-independent version of InkChat to evaluate and demonstrate our ideas. This is built on top of the portable framework presented in [70], whose primary purpose is to collect digital ink across a variety of platforms and to provide a platform-independent, consistent interface for digital ink applications. As a result, InkChat is available on these platforms and can process ink data without knowing the underlying details. To support digital ink data portability, InkChat uses InkML to represent digital ink data as it provides digital ink streaming and archival support independent of platforms. This allows flexible interchange of digital ink data in collaborative environments and, in addition, allows cut and paste of digital ink between applications, e.g. between Microsoft Office 2010 and InkChat.

Figure 9.7 shows the user interface of InkChat. A number of control buttons are located at the top of the canvas and grouped together to minimize the distance pen movement. InkChat provides a set of pre-defined colors and a palette to create new colors, if needed. To accommodate the needs of different writing activities, we have developed a few brush types. For example, one can choose the pen or pencil for diagramming and the tear drop brush for digital painting or calligraphy. Editing is also supported. This includes redo, undo, and select, cut, copy, and paste of different kinds of content, such as images, typed text, and digital ink.

10.2 InkChat Support for Multimodality

InkChat is a multi-year ongoing project. Its primary design objective is to enhance mathematical collaboration by incorporating multiple modalities. This allows participants to flexibly choose the input methods that are most suitable for a particular topic. Below we describe the modalities that are supported by InkChat.

Ink Traces Handwriting is one of the most natural ways to input mathematics, as most mathematical notations are two dimensional, with elements of both writing and drawing. InkChat captures handwriting as ink traces and exchanges the data with other clients using InkML.

Voice InkChat also supports voice input. The voice stream is paired with the ink stream to improve the efficiency of collaboration. For example, one can verbally explain the underlying meaning of a complex diagram while drawing it on a shared canvas. This avoids the clutter and confusion that may arise when either input method is used individually.

Floating Pointer To support collaboration, InkChat provides users with floating pointers that can be used to point at target objects on the shared canvas without leaving any ink mark. Together with the voice channel, participants can point to and discuss aspects of the common canvas.

10.3 InkChat Support for Collaboration

10.3.1 Communication

The primary goal of InkChat is to allow users on different computers to collaborate on a shared canvas. It currently uses the popular Skype and Google Talk services for the communication channel, but other transport mechanisms could be used. The primary design principle is to give users the freedom to choose the communication mechanism without too much configuration. If one service is not available in a particular location, it is easy to switch to another. Conference mode is supported, where more than two participants can be involved in one conversation. Depending on the chosen underlying communication service, InkChat adopts different mechanism to exchange data with other participants. For example, when a P2P backbone is used, the conference is initiated by the host that has a connection with every other participant. Digital ink routing shares the same mechanism as audio routing, each piece of ink stroke will be broadcast by the host to all participants except the initiator.

10.3.2 Page Navigation

InkChat also supports page navigation. This has been found useful when participants wish to cover multiple topics in one session or to load previous work in the middle of a conversation. In both cases, the current page will first be saved to the file system as an InkML file. Then the Ink Canvas will send a page request to the file system to check if the next page already exists. If so, the Ink Canvas will parse the InkML file and load the content so that users can continue to work on that page. Otherwise, a blank page will be created. Figure 10.1 illustrates the communications used in page navigation.

10.3.3 Ink Editing

In collaboration it is useful to edit or modify a work in progress, and in order to edit, it is necessary to be able to erase digital ink. InkChat provides two ways to erase ink: either by erasing whole strokes or parts of strokes. We call these “stroke-wise” and “point-wise” erasing. Stroke-wise erasing uses a hit testing method to detect whether

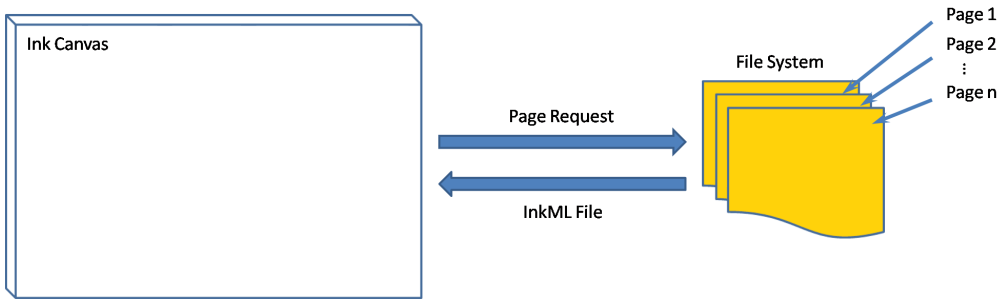


Figure 10.1: InkChat page model

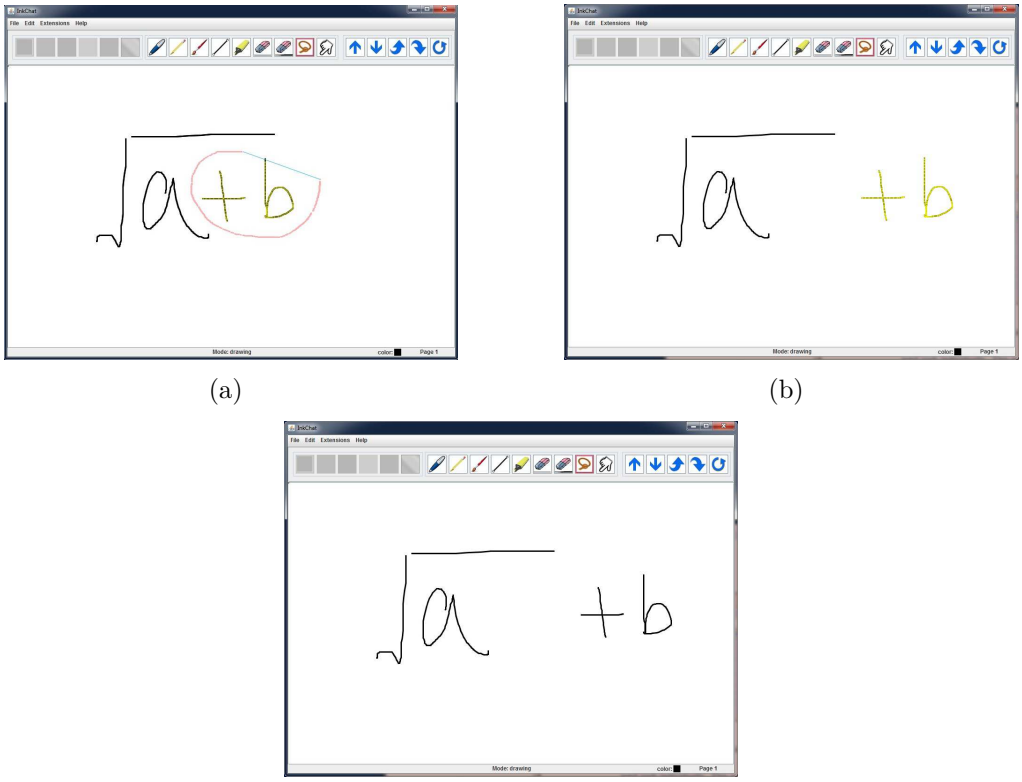
a particular stroke is selected. If so, it removes the stroke from the canvas and re-renders other strokes that may be affected. Point-wise erasing erases part of a stroke instead of removing the whole from the canvas. A stroke may be split into pieces when using point-wise erasing, and this requires the application to detect where the stroke is broken up. Point-wise erasing uses a hit testing method as well, and this returns a collection of ink points that need to be removed from the target stroke. It then groups the remaining ink points into new strokes and calculates the properties for each, including starting time and duration. The new strokes are then placed in sequence by starting time.

10.3.4 Drag and Drop

InkChat allows existing ink to be moved on the canvas using drag and drop. This uses a special lasso cursor to select the content to be moved. This lasso is a free selection tool that allows users to create a selection by encircling a region with a pen. The Lasso is useful in mathematical domains as users may often wish to select a portion of a mathematical expression. Figure 10.2 shows an example of using Lasso. Notably, as the bounding boxes of character “a” and “+” overlap, a rectangular selection is not suitable for this operation.

10.3.5 Real-Time Mirroring

InkChat is able to animate the drawing of ink strokes, and uses this to render the ink of collaborators as it is being written. To avoid jarring and distraction that are caused by large visual changes, InkChat splits long ink traces into small pieces and



(a) (b)

(c)

Figure 10.2: An example of using Lasso in InkChat. (a) Selection (b) Drag (c) Drop

send each individually. This is to allow smooth rendering on each participant's canvas. The representation of digital ink data is the key to the success of this animation. Collaboration sessions often take place in heterogeneous environments, where participants may work on different platforms and use various pen devices. These pen devices typically have different settings such as sampling rate, sensitivity, channel properties and so on, and consequently output digital ink data with different characteristics. This requires digital ink data to be represented in a flexible, platform- and vendor-independent format so that the animation is possible across different platforms. Meanwhile, ink strokes must be organized in time order in order to support smooth rendering and synchronization with other modalities.

We have found InkML suitable for these animation purposes. It is platform- and vendor-independent and allows complete and accurate representation of digital ink by capturing and recording information such as the device characteristics, pen tilt, pen pressure and so on. Most importantly, it provides a wide range of features to support smooth rendering and synchronization.

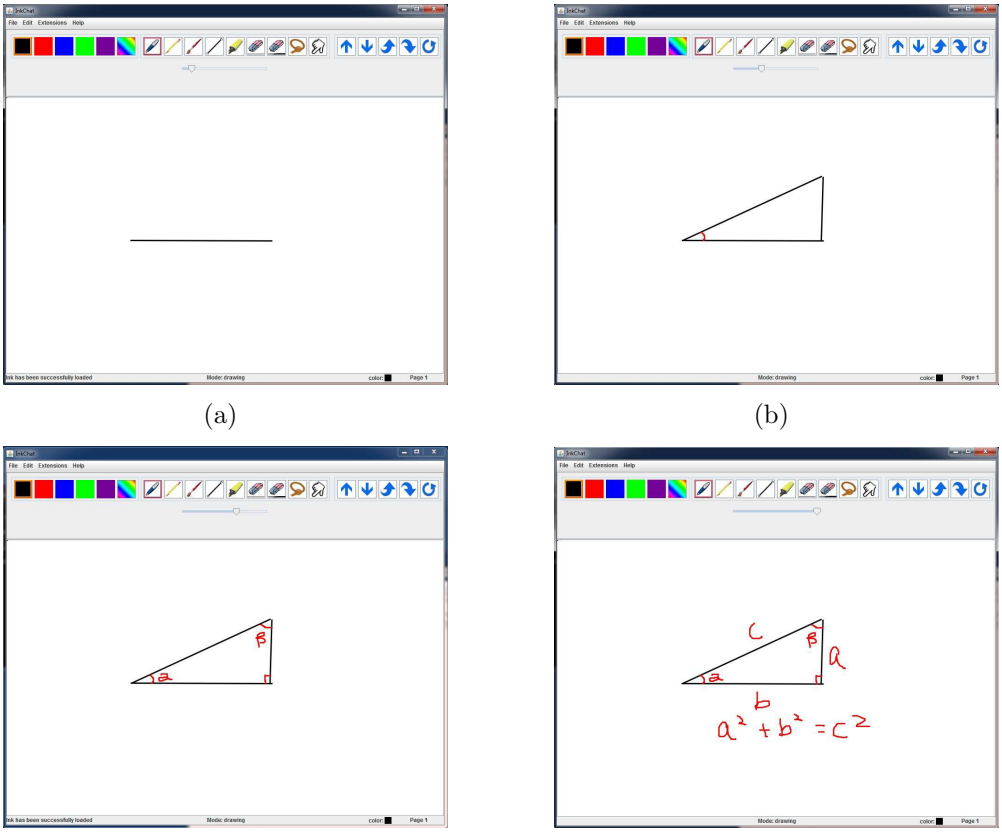


Figure 10.3: InkChat animation.

10.3.6 Session Recording and Playback

Collaboration sessions may be recorded and stored for later playback, analysis or annotation. InkChat stores digital ink in InkML archival style which keeps the contextual information and ink traces separately in order to achieve compact representation. When playback is desired, the digital ink data can be efficiently converted into streaming style which organizes ink strokes along with contextual information in time order [80]. In addition, each ink trace and its constituent ink points can be timestamped in order to support accurate synchronization with content input by other modalities. Figure 10.3 shows an example of playback in InkChat.

10.4 Summary

We have shown how InkChat has been made to support multimodal interaction and communication. We have found, informally, that these features greatly enhance the InkChat's effectiveness for collaboration. This seems to arise primarily by separating the creation and manipulation of the objects of discourse (diagrams, equations, and so on) from the discussion about the objects and the manipulations. It is an ongoing question of investigation to quantify these findings.

We would like thank Michael Friesen, Vadim Mazalov, Amit Regmi, and Coby Viner for their contributions to the implementation of InkChat. We would also like to thank James Wake for investigating how InkChat may be integrated in other environments, including Google Hangouts.

Chapter 11

Conclusion

Digital ink technology has evolved over the years. Its use in form filling, free-hand input, document annotation and collaboration has been well recognized. Pen input today is almost everywhere, from large whiteboard systems to small PDAs, from portable Tablet PCs to smart phones. A number of well-known digital ink applications have been developed on these platforms. While considerable progress has been achieved, digital ink technology can still be substantially improved. The primary objective of this thesis was to identify, investigate and solve key problems in developing effective digital ink applications. To achieve this goal, we have investigated three important aspects of digital ink in the broad areas of representation, recognition, and collaboration. The presented research results present new techniques to obtain compact digital ink for efficient data processing, and also present new methods to identify features in handwritten characters. In addition, we have demonstrated how a collaboration framework can be organized to take advantage of these ideas. The techniques presented may be applied in a number of domains, including mathematics, which is our motivating use case. In particular, we made the contributions aligned below.

We have presented an algorithm that can precisely approximate a digital ink curve through selecting a subset of points from the original trace. The algorithm can be used to compact the representation of digital ink while preserve the shape of curves, which allows efficient processing, transmission and storage of digital ink. We have also examined the suitability of InkML to represent sophisticated digital ink. To demonstrate our ideas, we have presented two brush models, Round Brush

and Tear Drop Brush. Both brush models can be used to support different writing activities and to improve the rendering quality of digital ink. The Tear Drop Brush has been adopted into InkML at our suggestion. Last, we have described a data binding solution for two-way conversion between SketchML and InkML, making it possible to exchange digital ink data between the two commonly used formats.

To help with recognition, we have presented an algorithm that can identify automatically the determining points in handwritten mathematical symbols. Knowing the determining points of each symbol can help us improve two-dimensional mathematical recognition. In contrast to existing methods, which treat digital ink traces as a collection of discrete points, this algorithm relies on interpreting ink traces as single points in a functional space. This allows device independence, on one hand, and a simple formulation of homotopic deformation, on the other. To evaluate the performance of the algorithm, we have tested it against a database of handwritten mathematical characters. The experiments showed promising results. We have also learned that by using the homotopic method, one can derive determining points in those samples that are significantly from the reference symbol with high accuracy. So we further studied the factors to be taken into account in performing such homotopies. We examined two strategies for possible starting points for the homotopy, and the relation between the distance from the starting points to the ending points and the number of steps required. The first strategy performs a homotopy to the average symbol, constructed from all samples of the same type. The second strategy uses a homotopy to the nearest neighbor with known determining points. We have presented our experimental results against a database of handwritten mathematical characters. The results suggested improved strategies to find determining points for poorly written characters.

To support digital ink collaboration, we have proposed a portable framework that can handle the details of a variety of platforms and provide a consistent, platform-independent interface for digital ink applications. Applications that are built on top of the framework can collect digital ink across all the supported platforms without knowing their underlying details. In addition, it adopts InkML to support digital ink collaboration in heterogeneous environments. We have also analyzed the design issues in incorporating multimodal interactions in collaboration. Moreover, we have described a streaming digital ink framework for multi-user online collaboration in a pen-based and graphical environment. This environment allows participants conducting and archiving collaborative sessions that involve synchronized voice and

digital ink on a shared canvas. The digital ink is represented as InkML, allowing special recognizers for different content types, such as mathematics and diagrams. The collaborative sessions may be recorded and stored for later playback, analysis or annotation. To demonstrate our ideas, we have presented InkChat, a whiteboard application, which can be used to conduct collaborative sessions on a shared canvas. It is portable across a variety of platforms and allows participants to use voice and digital ink independently and simultaneously. InkChat has been found useful in mathematical collaboration.

Bibliography

[1] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Evaluation of multimodal input for entering mathematical equations on the computer. In *Proceedings of the 2005 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1184–1187. ACM, 2005.

[2] MIT Design Rationale Group. MIT SketchML Format. http://rationale.csail.mit.edu/ETCHASketches/format/index.html.

[3] Stephen M. Watt and Tom Underhill (Editors). Ink Markup Language (InkML) W3C Recommendation. http://www.w3.org/TR/InkML/, September 2011.

[4] Isabella Guyon. Unipen 1.0 Format Definition. http://www.unipen.org/dataformats.html, 1992.

[5] Isabelle Guyon, Lambert Schomaker, Réjean Plamondon, Mark Liberman, and Stan Janet. UNIPEN Project of On-line Data Exchange and Recognizer Benchmarks. In *Proceedings of the 12th International Conference on Pattern Recognition (IAPR-IEEE)*, pages 29–33, 1994.

[6] Slate Corporation. JOT - A Specification for an Ink Storage and Interchange Format. http://unipen.nici.kun.nl/jot.html, 1996.

[7] Stephen M. Watt. Polynomial Approximation in Handwriting Recognition. In *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation*, SNC '11, pages 3–7. ACM, 2011.

[8] Oleg Golubitsky and Stephen M. Watt. Online Stroke Modeling for Handwriting Recognition. In *Proceedings of the 18th Annual International Conference on Computer Science and Software Engineering*, CASCON '08, pages 72–80. ACM, 2008.

[9] Bruce W. Char and Stephen M. Watt. Representing and Characterizing Handwritten Mathematical Symbols through Succinct Functional Approximation. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, ICDAR '07, pages 1198–1202. IEEE Computer Society, 2007.

[10] Oleg Golubitsky and Stephen M. Watt. Distance-Based Classification of Handwritten Symbols. *International Journal on Document Analysis and Recognition*, 13(2):133–146, June 2010.

[11] Elena Smirnova and Stephen M. Watt. A Context for Pen-Based Mathematical Computing. In *Proceedings of the 2005 Maple Summer Workshop*, pages 409–422, Waterloo, Canada, 2005.

[12] Sargur Srihari, Chen Huang, and Harish Srinivasan. On the Discriminability of the Handwriting of Twins. *Journal of Forensic Sciences*, 53:430–446, 2008.

[13] Maplesoft Inc. Maple user manual.

[14] Microsoft Inc. OneNote 2010.

[15] Iotum Inc. Calliflower. `http://www.calliflower.com/`.

[16] Dabbleboard Inc. Dabbleboard. `http://www.dabbleboard.com`.

[17] Rui Hu and Stephen M. Watt. Optimization of Point Selection on Digital Ink Curves. In *Proceedings of the 2012 International Conference on Frontiers in Handwriting Recognition*, pages 525–530, September 2012.

[18] James George Dunham. Optimum Uniform Piecewise Linear Approximation of Planar Curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1):67–75, January 1986.

[19] John Albert Horst and Isabel Beichl. Efficient Piecewise Linear Approximation of Space Curves Using Chord and Arc Length. In *Proceedings of the SME Applied Machine Vision*, pages 1–12, 1996.

[20] Zicheng Liu, Henrique S. Malvar, and Zhengyou Zhang. System and Method for Ink or Handwriting Compression. *United States Patent No US 7,302,106 B2*, November 2007.

[21] Vadim Mazalov and Stephen M. Watt. Linear Compression of Digital Ink via Point Selection. In *10th IAPR International Workshop on Document Analysis Systems*, DAS 2012, pages 429–434, 2012.

[22] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[23] Joseph J. LaViola. Symbol Recognition Dataset. http://pen.cs.brown.edu/symbolRecognitionDataset.zip.

[24] Rui Hu. Portable Implementation of Digital Ink: Collaboration and Calligraphy. Master's thesis, The University of Western Ontario, London, Canada, 2009.

[25] Xizhi Wang. Orchid Pavilion Gathering. `http://en.wikipedia.org/wiki/Orchid_Pavilion_Gathering`.

[26] Steve Strassmann. Hairy brushes. In *Proceedings of the 1986 International Conference and Exhibition on Computer Graphics and Interactive Techniques*, SIGGRAPH'86, pages 225–232, 1986.

[27] Helena T.F. Wong and Horace H.S Ip. Virtual Brush: A Model-Based Synthesis of Chinese Calligraphy. *Computers & Graphics*, 24(1):99 – 113, 2000.

[28] Rui Hu and Stephen M. Watt. InkChat: A Collaboration Tool for Mathematics. In *Electronic Proceedings of the 2013 Workshop on Mathematical User Interface*, MathUI'13, 2013.

[29] Microsoft Inc. Ink Serialized Format (ISF) Specification.

[30] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson (editors). Scalable Vector Graphics (SVG) 1.1 Specification.

[31] Rui Hu and Stephen M. Watt. From MIT SketchML to InkML, or There and Back Again. In *Proceedings of the 10th IAPR International Workshop on Document Analysis Systems Short Paper*, DAS 2012, pages 26–27. IEEE Computer Society, March 27-29, 2012.

[32] Birendra Keshari, Sriganesh Madhvanath, Manoj Prasad A, Muthuselvam Selvaraj, and Stephen M. Watt. Sharing Digital Ink in Heterogeneous Collaborative Environments. In *Proceedings of the IAPR International Conference on Frontiers in Handwriting Recognition (ICFHR 2008)*, pages 580–585, Montreal, QC, Canada, CENPARMI Concordia University, Aug 2008.

[33] Mike Oltmans, Christine Alvarado, and Randall Davis. ETCHA Sketches: Lessons Learned from Collecting Sketch Data. In *Proceedings of the Making Pen-Based Interaction Intelligent and Natural*, 2004.

[34] B. Keshari and S. Watt. Streaming-Archival InkML Conversion. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, pages 1253–1257, Washington, DC, USA, 2007. IEEE Computer Society.

[35] Rui Hu and Stephen M. Watt. Determining Points on Handwritten Mathematical Symbols. In *Proceedings of the 2013 Conferences on Intelligent Computer Mathematics*, pages 168–183, July 2013.

[36] Mario Pechwitz and Volker Märgner. Baseline Estimation for Arabic Handwritten Words. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition*, pages 479–484, 2002.

[37] María Teresa Infante Velázquez. Metrics and Neatening of Handwritten Characters. Master's thesis, The University of Western Ontario, London, Canada, 2010.

[38] Scott D. Connell and Anil K. Jain. Template-Based Online Character Recognition. *Pattern Recognition*, 34:1–14, 1999.

[39] Richard Zanibbi, Kevin Novins, James Arvo, and Katherine Zanibbi. Aiding manipulation of handwritten mathematical expressions through style-preserving morphs. In *Proceedings of Graphics Interface*, pages 127–134, 2001.

[40] Majid Harouni, Dzulkifli Mohamad, and Abdolreza Rasouli. Deductive Method for Recognition of On-Line Handwritten Persian/Arabic Characters. In *Proceedings of the 2nd International Conference on Computer and Automation Engineering*, volume 5 of *ICCAE'10*, pages 791–795, February 2010.

[41] Mark Anthony Armstrong. *Basic Topology.* Undergraduate Texts in Mathematics. Springer, 1979.

[42] Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Orientation-Independent Recognition of Handwritten Characters with Integral Invariants. In *Proceedings of Joint Conference of ASCM 2009 and MACIS 2009: Asian Symposium of Computer Mathematics and Mathematical Aspects of Computer and Information Sciences*, ASCM 2009, pages 252–261, 2009.

[43] Oleg Golubitsky, Vadim Mazalov, and Stephen M. Watt. Toward Affine Recognition of Handwritten Mathematical Characters. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, DAS '10, pages 35–42. ACM, 2010.

[44] Rui Hu and Stephen M. Watt. Identifying Features via Homotopy on Handwritten Mathematical Symbols. In *Proceedings of 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, SYNASC 2013, Timiŏara Romania, 2013. IEEE Computer Society.

[45] M. Koschinski, Hans-Jürgen Winkler, and Manfred Lang. Segmentation and Recognition of Symbols within Handwritten Mathematical Expressions. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2439–2442, Detroit, USA, 1995.

[46] Elena Smirnova and Stephen M. Watt. Aspects of Mathematical Expression Analysis in Arabic Handwriting. In *Proceedings of the International Conference on Document Analysis and Recognition*, volume 2, pages 1183–1187, Curitiba, Brazil, 2007.

[47] Richard Zanibbi and Dorothea Blostein. Recognition and Retrieval of Mathematical Expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.

[48] Kang Kim, Taik-Heon Rhee, Jae Seung Lee, and Jin Hyung Kim. Utilizing Consistency Context for Handwritten Mathematical Expression Recognition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 1051–1055, 2009.

[49] Badr Al-Badr and Sabri A. Mahmoud. Survey and Bibliography of Arabic Optical Text Recognition. *Signal Processing*, 41(1):49–77, 1995.

[50] Adnan Amin. Off-Line Arabic Character Recognition: the State of the Art. *Pattern Recognition*, 31(5):517–530, 1998.

[51] Claudie Faure and Zi Xiong Wang. Structural Analysis of Handwritten Mathematical Expressions. In *Proceedings of the 9th International Conference on Pattern Recognition*, volume 1, pages 32–34, 1988.

[52] Ernesto Tapia and Raúl Rojas. Recognition of Online Handwritten Mathematical Expressions Using a Minimum Spanning Tree Construction and Symbol Dominance. In *Proceedings of Graphics Recognition.*, volume 3088, pages 329–340. Springer, 2004.

[53] Chuanjun Li, Robert Zeleznik, Timothy Miller, and Joseph J. LaViola Jr. Online Recognition of Handwritten Mathematical Expressions with Support for Matrices. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4, 2008.

[54] Oleg Golubitsky and Stephen M. Watt. Online Recognition of Multi-Stroke Symbols with Orthogonal Series. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, ICDAR'09, pages 1265–1269, Barcelona, Spain, 2009.

[55] Xiaojie Wu. Achieving Interoperability of Pen Computing with Heterogeneous Devices and Digital Ink Formats. Master's thesis, The University of Western Ontario, London, Canada, 2004.

[56] Amit Regmi. Supporting Multimodal Collaboration with Digital Ink and Audio. Master's thesis, The University of Western Ontario, London, Canada, 2009.

[57] Hai Ning, John R. Williams, Alexander H. Slocum, and Abel Sanchez. InkBoard - Tablet PC Enabled Design oriented Learning. In *Proceedings of the 7th IASTED International Conference on Computers and Advanced Technology in Education*, CATE 2004, pages 154–160. ACTA Press, August 16-18, 2004.

[58] Jay Beavers, Tim Chou, Randy Hinrichs, Chris Moffatt, Michel Pahud, Lynn Powers, and Jason Van Eaton. The Learning Experience Project: Enabling Collaborative Learning with ConferenceXP. Technical Report MSR-TR-2004-42, Microsoft Research, 2004.

[59] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Evaluation of multimodal input for entering mathematical equations on the computer. In *Proceedings of the CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1184–1187. ACM, 2005.

[60] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow. QuickSet: Multimodal Interaction for

Distributed Applications. In *Proceedings of the fifth ACM international conference on Multimedia*, MULTIMEDIA '97, pages 31–40. ACM, 1997.

[61] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. LiveBoard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI'92, pages 599–607, New York, NY, USA, 1992.

[62] Elin R. Pedersen, Kim McCall, Thomas P. Moran, and Frank G. Halasz. Tivoli: an electronic whiteboard for informal workgroup meetings. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 391–398, New York, NY, USA, 1993. ACM.

[63] Gregory D. Abowd, Jason Brotherton, and Janak Bhalodia. Classroom 2000: A system for capturing and accessing multimedia classroom experiences. In *Proceedings of 1998 Conference Summary on Human Factors in Computing Systems*, CHI'97, pages 20–21. ACM, 1998.

[64] Apple Inc. Inkwell.

[65] Gerald Friedland, Lars Knipping, Raúl Rojas, and Ernesto Tapia. Teaching with an intelligent electronic chalkboard. In *ETP'04: Proceedings of the 2004 ACM SIGMM workshop on Effective telepresence*, pages 16–23. ACM, 2004.

[66] Microsoft Inc. Microsoft Windows XP Tablet PC Edition Software Development Kit.

[67] Inkscape Community. Inkscape. `http://www.inkscape.org`.

[68] Jon Ferraiolo, Fujisawa Jun, and Dean Jackson(editors). Scalable Vector Graphics (SVG) 1.1 Specification. `http://www.w3.org/TR/SVG/`.

[69] Amit Regmi and Stephen M. Watt. A Collaborative Interface for Multimodal Ink and Audio Documents. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 901–905, 2009.

[70] Rui Hu, Vadim Mazalov, and Stephen M. Watt. A Streaming Digital Ink Framework for Multi-Party Collaboration. In *Proceedings of the 2012 Conference on Intelligent Computer Mathematics*, CICM'12, pages 81–95, 2012.

[71] Timothy Gowers and Michael Nielsen. Massively Collaborative Mathematics. *Nature*, 461:879–881, 2009.

[72] Microsoft Inc. Skype. `http://www.skype.com/`.

[73] Google Inc. Google Talk. `http://www.google.com/talk/`.

[74] Amit Regmi and Stephen M. Watt. A Collaborative Interface for Multimodal Ink and Audio Documents. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, ICDAR 2009, pages 901–905, 2009.

[75] Vadim Mazalov and Stephen M. Watt. Writing on Clouds. In *Proceedings of the 2012 Conferences on Intelligent Computer Mathematics*, CICM 2012. Springer Verlang, July 9-14 2012.

[76] Vadim Mazalov and Stephen M. Watt. Digital Ink Compression via Functional Approximation. In *Proceedings of 12th International Conference on Frontiers in Handwriting Recognition*, ICFHR 2010, pages 688–694, November 16-18, 2010.

[77] Vadim Mazalov and Stephen M. Watt. Linear Compression of Digital Ink via Selection of Subsets of Points. In *Proceedings of the 10th IAPR International Workshop on Document Analysis Systems*, DAS 2012, pages 429–434, March 27-29, 2012.

[78] Stephen M. Watt. On the Mathematics of Calligraphy (Invited Talk), 2009. International Conference On Mathematics Mechanization – In honor of Professor Wen-Tsun Wu's ninetieth birthday.

[79] Elizabeth Burd, Dawn Overy, and Ady Wheetman. Evaluating Using Animation to Improve Understanding of Sequence Diagrams. In *Proceedings of the 10th International Workshop on Program Comprehension*, IWPC 2002, page 107. IEEE Computer Society, 2002.

[80] Birendra Keshari and Stephen M. Watt. Streaming-Archival InkML Conversion. In *Proceedings of the 2007 International Conference on Document Analysis and Recognition, (ICDAR 2007)*, pages pp. 1253–1257, Curitiba, Brazil, September 23-26 2007.

Curriculum Vitae

Name

Rui Hu

Post-secondary Education

The University of Western Ontario
London, Ontario, Canada
PhD (Computer Science) 2010 - 2013
Supervisor: Dr. *Stephen M. Watt*

The University of Western Ontario
London, Ontario, Canada
MSc. (Computer Science) 2008 - 2009
Supervisor: Dr. *Stephen M. Watt*

Xidian University
Xi'an, China
Graduate study. 2007 - 2008
Supervisor: Dr. *Qingshan Li*

Xidian University
Xi'an, China
B. E. 2003 - 2007

Work Experience

Research and Development Specialist
Embium (formerly Cyborg Trading Systems)
Project: Advanced Financial Market Simulation
based on Intelligent Agents and Cloud Technology
2011 - 2013

Teaching Assistant
The University of Western Ontario
2008 - 2013

Research Assistant
Ontario Research Centre for Computer Algebra
2008 - 2013

Awards

- Student Grant in Conferences on Intelligent Computer Mathematics, 2013
- Mitacs Accelerate Scholarship, 2012
- Most Valuable Player on Western Men's Table Tennis Team, 2012
- Student Grant in Conferences on Intelligent Computer Mathematics, 2012
- First Prize in UWO Research in Computer Science, 2010
- Western Graduate Research Scholarship, 2008-2013
- Dean Scholarship of Software Engineering School, Xidian University, 2005-2007
- University Scholarship, Xidian University, 2003-2007

Publications

1. Rui Hu and Stephen M. Watt. *Identifying Features via Homotopy on Handwritten Mathematical Symbols*. Proc. 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, (SYNASC 2013), September 23-26 2013, Timişoara, Romania, IEEE Computer Society (to appear).

2. Rui Hu and Stephen M. Watt. *Determining Points on Handwritten Mathematical Symbols*. Proc. 2013 Conferences on Intelligent Computer Mathematics, (CICM 2013), pages 168-183, July 8-12 2013, Bath, UK, Springer Verlag LNAI 7961.

3. Rui Hu and Stephen M. Watt. *InkChat: A Collaboration Tool for Mathematics*. Electronic Proc. 2013 Workshop on Mathematical User Interfaces, (MATHUI 2013), July 10 2013, Bath, UK. `http://cermat.org/events/MathUI/13`

4. Rui Hu and Stephen M. Watt. *Optimization of Point Selection on Digital Ink Curves*. Proc. 13th International Conference on Frontiers in Handwriting Recognition, (ICFHR 2012), pages 525-530, September 18-20 2012, Bari, Italy, IEEE Computer Society.

5. Rui Hu, Vadim Mazalov, and Stephen M. Watt. *A Streaming Digital Ink Framework for Muiti-Party Collaboration*. Proc. 2012 Conferences on Intelligent Computer Mathematics, (CICM 2012), pages 81-95, July 9-14 2012, Bremen, Germany, Springer Verlang LNAI 7362

6. Rui Hu and Stephen M. Watt. *From MIT SketchML to InkML, or There and Back Again*. Proc. 10th IAPR International Workshop on Document Analysis Systems Short Papers, (DAS 2012), pages 26-27, March 27-29 2012, Gold Coast, Australia, IAPR.

Conference Presentations

1. *Determining Points on Handwritten Mathematical Symbols*, 2013 Conferences on Intelligent Computer Mathematics, (CICM 2013), Bath, UK, July 11 2013.

2. *InkChat: A Collaboration Tool for Mathematics*, 2013 Workshop on Mathematical User Interfaces Workshop, (MathUI 2013), Bath, UK, July 10 2013.

3. *Pen-Based Collaboration Tools for Mathematics*, Doctoral Programme in 2013 Conferences on Intelligent Computer Mathematics, Bath, UK, July 10 2013.

4. *A Streaming Digital Ink Framework for Muiti-Party Collaboration*, 2012 Conferences on Intelligent Computer Mathematics, (CICM 2012), Bremen, Germany, July 9, 2012

5. *Writing on Clouds*, 2012 Conferences on Intelligent Computer Mathematics, (CICM 2012), Bremen, Germany, July 9, 2012 (Presented for Vadim Mazalov and Stephen M. Watt)

6. *Pen-Based Collaboration Tools for Mathematics*, Doctoral Programme in 2012 Conferences on Intelligent Computer Mathematics, Bremen, Germany, July 11 2012.

7. *Portable Capture and Rendering of Calligraphic Ink Strokes*, 2009 Workshop on Pen-Based Mathematical Computation, (PenMath), Grand Bend, Canada, July 8 2009.