Supporting Mathematical Handwriting Recognition through an Extended Digital Ink Framework

(Spine title: Supporting Mathematical Handwriting Recognition with Digital Ink)
(Thesis format: Monograph)

By
Kevin Durdle

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
University of Western Ontario
London, Ontario, Canada

Certificate of Examination

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

Supervisor	Examining Board
______________________________	______________________________
Supervisory Committee	______________________________
______________________________	______________________________

The thesis by

Kevin James Durdle

entitled:

Supporting Mathematical Handwriting Recognition through an Extended Digital Ink Framework

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date__________________________ ______________________________
Chair of the Thesis Examination Board

Abstract

The success rates of recognizing mathematical and other visual languages have fallen significantly behind those of textual language recognizers. The failure of mathematical recognizers is attributed to the many difficulties encountered during recognition, resulting in unacceptably low rates of accuracy. By creating a framework that focuses on supporting mathematics with digital ink, it is expected that the improved development environment will contribute to the flexibility and successfulness of future mathematical handwriting recognizers.

In creating such a mathematical framework, the following factors influencing recognition are examined: hardware and software requirements; digital ink requirements; properties of mathematical expressions; differences of visual and textual languages; methods of recognition; user interface requirements; and our contribution, the requirements of a development environment friendly to mathematics which supports digital ink. Using these guidelines, the success rates of future work in recognizing handwritten mathematics should improve, further enhancing this emerging technology.

Keywords

Recognition of handwritten mathematics; Portable Digital Ink Architecture; Digital ink; Stylus; Tablet PC; Pocket PC

Acknowledgements

I want to especially thank my fiancée and soon to be wife Amy, for her intense editing and critical analysis of this document. Without her support, this would have been a very difficult assignment. Furthermore, this thesis would have been impossible without the help of my supervisor, Dr. Stephen Watt. Special thanks to the ORCCA Pen Group including Clare So, Elena Sminovia, Xiaofang Xie and Xiaojie Wu for their assistance in many places. There are other Professors, students and colleges who also helped in various ways with my M.Sc., it would be impossible to list them all. Lastly, thank you to my parents and family for encouragement and your non-technical assistance with my M.Sc.

Table of Contents

List of Tables

List of Figures

List of Appendices

Glossary

API: An acronym for ***A***pplication ***P***rogramming ***I***nterface. APIs allow developers to create applications that interact with previously provided functionalities.

Digital Ink, Ink: An electronic or digital representation of ink. Displays that support input, typically through touch screens or electromagnetically, are capable simulating the use of a pen, by leaving behind a digital ink representation.

DPI: An acronym for ***D***ots (or pixels) ***P***er ***I***nch.

OCR: An acronym for ***O***ptical ***C***haracter ***R***ecognition, it is a technology reads text from paper and translates the image into a format that permits manipulation by a computer.

Offline Recognition: A form of OCR, offline recognition is used to translate digital ink into text at a later date, often when additional input is non possible or convenient. The IBM Crosspad used this technology; after accepting input, users would then process the input as a separate process.

Online Recognition: A technology that permits the immediate translation of digital ink into text, as input is being accepted. The Tablet PC uses this technology.

PDA: An acronym for ***P***ersonal ***D***igital ***A***ssistant, these handheld devices include Palm Pilots and Pocket PCs are palm sized, and offer an assortment of features including calendars, address books, email access and multimedia streaming.

PDIA: An acronym for ***P***ortable ***D***igital ***I***nk ***A***rchitecture, PDIA is a solution which provides a means of using digital ink from one device or platform on another, without having to worry about conversions of data formats.

Textual Language: Languages that rely on the use of discrete characters to portray all information that is desired to be displayed.

Visual Language: Languages that rely on the use of two or more dimensions to portray meaning. Mathematics and music are two examples, both convey information by the placement of symbols relative to other symbols.

Chapter 1 Introduction

1.1 Overview

For millennia people have used diagrams as a means of preserving ideas or communicating with others: Native societies used symbolic drawings, the Egyptians had hieroglyphs. By 3000 BCE the Babylonian and Egyptian cultures had advanced sufficiently enough to permit the study of mathematics to aid with practical affairs. As mathematics evolved in complexity, the adoption of a two-dimensional format was a natural means of communicating as efficiently as possible.

The use of two-dimensional notation is still used in mathematics, in addition to various other fields including engineering, chemical and biological sciences, and music, among others. In these disciplines the ability to use visual aids to communicate may permit a better understanding of someone's intentions than is possible with a linear formed grammar. This is because visual languages allow for a simpler and more concise expression using fewer symbols than is possible with textual languages.

The introduction of the computer introduced the first major changes in documents since the printing press, almost five centuries earlier. Initially, computers supported the input of small character sets of approximately 64 characters and then 256 ASCII characters, enough for most Western European languages. Today, 16 bit character codes enable us to represent 65,536 characters, providing support for symbols from most of the world's languages. Even if the number of characters supported by computers were large enough to support all characters in two-dimensional languages, they would become impossibly difficult to enter. The result is that these languages continue to rely on textual representations if one is to enter them into a computer.

Using current technologies, computers are capable of manipulating, processing and displaying mathematics, along with many other two-dimensional languages. However the task of entering the required and often non-intuitive notations has been the responsibility of the user, as reliable recognition processes do not exist that support generalized inputs.

Unlike mathematics and other two-dimensional languages, automatic recognition of textual information has benefited from significant amounts of research since the early 1960s [1, 2]. From the 1960s through to the 1980s, technological improvements permitted software to produce natural language handwriting recognition applications for the first time. Over the past two decades, character recognizers using both online and offline methodologies have advanced enough to enable users to *write* input with a stylus or pen based device, instead of *entering* their input through a keyboard. Furthermore, Optical Character Recognizer applications have enabled users to convert paper documents to electronic formats with scanner technologies, avoiding the use of keyboards altogether.

Although research in structure analysis of two-dimensional patterns coincided with textual recognition in the early 1960s [3], the advances seen in string recognition have not been observed in mathematical or two-dimensional recognition. As noted by other authors [4, 5, 6, 7], the degree of complexity of structural analyzing combined with the required CPU power, prevented considerable progress in mathematical recognition.

The 1980s and 1990s saw the introduction of personal computers with processors powerful enough to permit handwriting recognition. Unfortunately, the embracing of handwriting and digital ink by industry during this time was lethargic. When supported, available ink data formats were proprietary and device dependent, recognition accuracies were low and vendor support was short lived.

In 1993, Graffiti was introduced by Palm Corporation. Similar to the “Unistroke Symbols” research originally by Xerox, Graffiti takes advantage of a limited character domain, requiring users to learn a special alphabet where each character can be drawn in a unique manner, without lifting the stylus from the device. The results are accuracy rates that are near perfect for experienced users. Since this time, natural language handwriting recognition has continued to improve steadily, whereas the recognition of mathematics has not received similar attention.

It is our belief that by taking advantage of previous research results - today's powerful processors, hardware improvements and the recognition of a pen as a valid input device - a unique opportunity arises that necessitates revisiting the creation of a mathematical handwriting recognition application. By relieving the user of the burden of translation from mathematical notation to ASCII text, a mathematics recognition system would enhance the usefulness of computers as a tool for mathematics and document handling.

1.2 Demand for a Mathematical Recognizer

From a general commercial point of view, investment in digital ink is synonymous with risk and isolated market opportunities. There is no single dominant platform or even a device to target: customers who use digital ink want both ultra mobility (PDA's and Cellular Phones) as well as processing power (Tablets and laptops). These hardware options, when combined with a selection of operating systems, result in further proprietary development models, inconsistent APIs, ink formats and fragmented marketing opportunities.

The creation of a mathematical handwriting recognition, end-to-end experience is a vision driven by researchers at the Ontario Research Centre for Computer Algebra (ORCCA) research lab. As this vision progressed, the topics within this thesis document emerged. Also emerging was an understanding that a math ink framework must be created to allow the successful implementation of a math recognizer.

1.3 Thesis Introduction

The research presented in this thesis supports the following:

> *The creation of a framework that supports digital ink for mathematics is necessary for constructing a flexible and successful mathematical handwriting recognition engine.*

By *framework* we mean the necessary foundations and supporting elements that are necessary for a math recognizer to succeed. Each chapter of this document refers to a specific element of this framework. By *flexible* we refer to the modular architecture that is detailed within this thesis. *Successful* indicates our belief that forthcoming chapters detail elements beyond the mathematical recognizer that are necessary for general adoption of any recognizer.

1.4 Thesis: Objective

The creation of a mathematical handwriting recognizer is a nontrivial task; most prototypes make such insurmountable assumptions that functional applications are never produced. Examples of such assumptions have included: *assuming* segmentation is completed, *assuming* character recognition is perfect and *assuming* context is previously define, among others. As part of a greater vision at ORCCA is the anticipation of a complete mathematical handwriting recognition engine, of which this thesis's objectives assists twofold: First to identify areas where previous researchers had made assumptions and second to create a unified digital ink interface.

Each of the assumptions addressed in this thesis will present either conclusions from my research as well as others, or introduce a much larger topic that are addressed by other members of the ORCCA research group. For instance, it is possible to draw from conclusions in areas such as hardware or software requirements without significantly increasing the scope of the thesis. In contrast, an entire thesis could be dedicated to the topic of creating a "mathematical dictionary" alone as there are enough intrinsic details. In these cases only an introduction and outline of requirements or suggestions are presented here to show how the works are related.

Secondly, we aim to cover in detail the creation of unified digital ink architecture. This unified architecture will assist in the success of future recognizers that are produced by ORCCA. Furthermore, a detailed architecture is implemented and conclusions based on our experiments are presented in this document.

1.5 Thesis: Contributions

Each chapter presents discussions, which progress from hardware and software requirements, to inking requirements, to user interface requirements and finally the conclusions we present based on experiments.

1.5.1 Chapter 2 Review of Hardware & Software Platforms

This chapter describes existing hardware platforms that are suitable to operate a math recognition engine. The basic hardware requirements are identified as well as an introduction to the minimal requirements of each platform targeted by a mathematical recognizer. Minimal software requirements are also identified, targeted Operating Systems must provide the ability to capture Stylus Down, Stylus Move and Stylus Up events. With these events, it is possible to simulate the entire inking experience, which is necessary to ensure a unified interface is created.

1.5.2 Chapter 3 Attributes of a Mathematical Ink Recognition Engine

This chapter details each of the attributes identified as critical to the success of a mathematical handwriting recognition engine. Here we account for how we arrived at our conclusions as to why these attributes are essential.

1.5.3 Chapter 4 Portable Digital Ink Architecture

Detailing the architecture which will enable us to address each prerequisite as identified previously, this chapter illustrates how our inking solution satisfies the requirements. We present a novel solution based on C++, C# and Java that will enable third party developers to extend and add to our framework.

1.5.4 Chapter 5 Mathematical Expression Properties

Introducing the properties of mathematical expressions, this chapter illustrates many of the challenges that exist in recognizing mathematics. Also presented are fundamental differences between two-dimensional mathematical languages and text based string languages.

1.5.5 Chapter 6 Math Recognition Survey

Presenting the commonly practiced processes of industry leaders in addressing mathematical handwriting recognition, this chapter introduces ORCCA's vision of how we can expand and improve on these practices.

1.5.6 Chapter 7 User Interface Requirements

Acknowledging no recognizer will yield 100% accuracy, it is equally important to consider how the user will interact with an application to correct translation errors. Presented in this chapter are several methods of interacting with users that have proven effective.

1.6 Conclusions of Introduction

Evolving from a long history of simple two-dimensional diagrams used by early civilizations to ASCII renditions of mathematics in computers today, there is a need to return mathematics to means of natural communication methods when interacting with computers. Continual improvements in both hardware and software have helped in recent years to improve handwriting recognition; it is our expectation that this thesis will further assist in overcoming the problems encountered in recognizing handwritten mathematical expressions.

Chapter 2 Review of Previous, Existing and Targeted Hardware and Software Platforms

2.1 Overview

After examining the potential of a mathematical recognizer, this chapter describes existing hardware platforms that are suitable for interoperating with a math recognition engine. Identified are the basic hardware requirements such as screen and input devices as well as an introduction to the minimal processing requirements of targeted platforms. Also identified are the minimal software requirements of these targeted platforms: targeted Operating Systems must support event based triggers, at a minimal *Stylus Down*, *Stylus Move* and *Stylus Up* events. Using these three events, it is possible to simulate the entire inking interface.

In presenting a history of mathematical tools from early devices to modern calculators, we illustrate our expectation of how a mathematical handwriting recognizer will become a key tool for mathematicians upon integration into industry supported applications such as Maple, Mathematica, etc.

2.2 A History Lesson

The use of hardware to support mathematics is not new; astronomical calendars have existed for nearly two millennia. The *Antikythera mechanism*, a mechanical device used by the Greeks circa 82AD [8] to aid in or possibly completely replace astronomical calendars illustrates the long lasting importance of external hardware supporting mathematics. Since the first commercial calculator[1] in 1954 [9], today's calculators are

[1] The IBM Type 601 was introduced in 1931, but was not commercially available. The Type 603 Electronic Multiplier was less of a calculator, more of a multiplier, capable of only multiplying two

capable of most all common numerical calculations and have displays that are used to display or plot results instantly.

Unlike the original calculators by IBM and others during the same era, dependence on dedicated hardware by mathematical tools and computers has become less common. The standardizations that occurred during the 1980s and 1990s have given increased power to computers and handheld devices, enabling software controlled devices.

Pen computing, or computing where a pen-like device is used as a primary means of input, is relatively new. Introduced in 1963, Ivan Sutherland documented one of the first non-theoretical pen-based computing solutions, the Sketchpad [10]. Although the theory of pen computing goes back to at least 1945 [53], it is only in the last decade that the use of a pen has become practical. This is understandable; it took until 1993 for Xerox's Unistroke Symbols and Palm's Graffiti to provide a widely accepted means of handwriting recognition, a feature that is expected in pen-based systems.

While textual or natural language handwriting recognition has improved steadily, the recognition of mathematics has only very recently become generally available with the introduction of xThink's MathJournal in July of 2004 [11]. Previous efforts at mathematical handwriting recognition include [12, 13, 14, 15, 37, 38, 39, 41, 42, 43, 45, 46, 49, 50, 51]; the majority of these have been academic pursuits with little commercial intent. While their product is capable of recognizing mathematics in limited domains, the MathJournal product is bound to the Tablet PC platform because of its reliance on the Tablet PC APIs. Relying on a single platform owned by Microsoft, it is unlikely xThink will expand its solutions to the larger scope of handheld devices.

As interactive displays become increasingly popular and varied, Palm Pilots, Pocket PCs, Tablet PCs and other pen-based computers are suitable replacements for many of today's calculators and are an obvious platform of choice for a math recognition engine. In existing software-based calculators, dedicated hardware has already become obsolete. It is only a matter of time before these soft calculators which resembled their hardware

numbers together. Only 100 were built; it was quickly replaced by the Type 604, which had the capability for addition, subtraction, multiplication, and division.

based predecessors, are replaced with intelligent digital whiteboards that are capable of freeform mathematical handwriting recognition, allowing researchers to focus on the problem at hand versus learning to use new tools.

2.3 Pencil, Paper and the Calculator versus Pen Computing

Sobel's 1973 article "Electronic Numbers" [16] describes the many advantages of using now primitive instruments with digital displays to aid various mathematical calculations. Today's calculators have improved significantly in many aspects, notably in usability and the ways results are displayed to users. Pen computing today can be compared to the circa 1973 or earlier stages of calculator development: it is not seen as necessary, deriving significant competition from basic tools, the pencil and paper.

Often overlooked, paper has always been a key instrument for mathematicians. To replace paper as a medium, a pen computing solution will need to be as simplistic as paper while offering additional benefits – just as the calculator had to offer additional benefits while justifying the exceptional cost. In comparison to an acceptable expense, in 1973 a scientific calculator cost approximately $500[1]. Today, accounting for inflation, these same calculators would cost $2200, as determined by the Bank of Canada inflation statistics, the price of a low-end Tablet PC. This expense is justifiable only if the features added will save the investor in other areas such as time, productivity, research, etc. The creation of a math recognition engine has the potential to combine the openness of paper with the power of complex math processing software, easily justifying the costs.

2.4 Digital Styli, Handhelds and Tablet PCs

Each device or platform, either palm- and pocket-sized PC or WAP-enabled mobile phone, each running its respective operating system, supports a well targeted paradigm regarding the purpose and applicability of mobile computing [17, 18]. With the

[1] In 1973 the actual price for a HP-80 Business Calculator was $395 US plus tax; a Victor Electronic 1800 Series Scientific calculator was $495, plus tax. The 1973 timeframe was used for consistency with Sobel's article "Electronic Numbers".

introduction of the Tablet PC in 2002, a new paradigm of mobile computing was introduced: fully functional, pen-based computing that offered a user interface designed for stylus input. A comprehensive timeline of computing advances leading to the release of the Tablet PC is available in [19, pages 9 - 17].

The Tablet PC is one hardware platform targeted by this thesis. Physically, they are comparable and often identical to laptops, often as powerful as desktop computers. Tablet PCs provide users with a fully functional, general purpose operating system, in this case Windows XP Tablet edition. Screen size on Tablet PCs range from 4 inches to 14 inches or larger, and can be oriented vertically or horizontally, providing users with the option to use that which they feel most natural with.

While Tablet PCs easily meet minimal standards of display size, memory and processing power, handheld computers represent a device classification that will be much closer to the necessary minimal requirement for a mathematical framework, as discussed in section 2.6. A larger concern with handheld devices is with their display size. Most current models of Palm Pilots support a 320 x 320 resolution screen, whereas Pocket PCs support a minimum 320 x 480, with some newer devices capable of 480 x 640. Both of these devices are oriented vertically, although Pocket PCs running the Windows Mobile 2003 operating system are capable of changing to a horizontally oriented screen layout, providing a more natural reading or user interface experience.

Another option is the use of a digital stylus, similar to the products that Wacom Technologies Company produce. For the purpose of this thesis, only digital styli that provide visual feedback are considered. As a stylus is only a peripheral device, assuming its input resolution is sufficient there should be no concerns about meeting minimal hardware standards.

2.5 Unacceptable Hardware Platforms

The importance of immediate feedback was recognized by Hayashi *et al* in 1968 [20] when they presented a system that was capable of recognizing, storing and processing

Chinese characters. Using a *lightpen,* users were able to draw the desired Chinese character on a grid by indicating the source and end vertices. After each pair of points was selected, the specified line was automatically drawn on the grid, as seen in Figure 2-1.

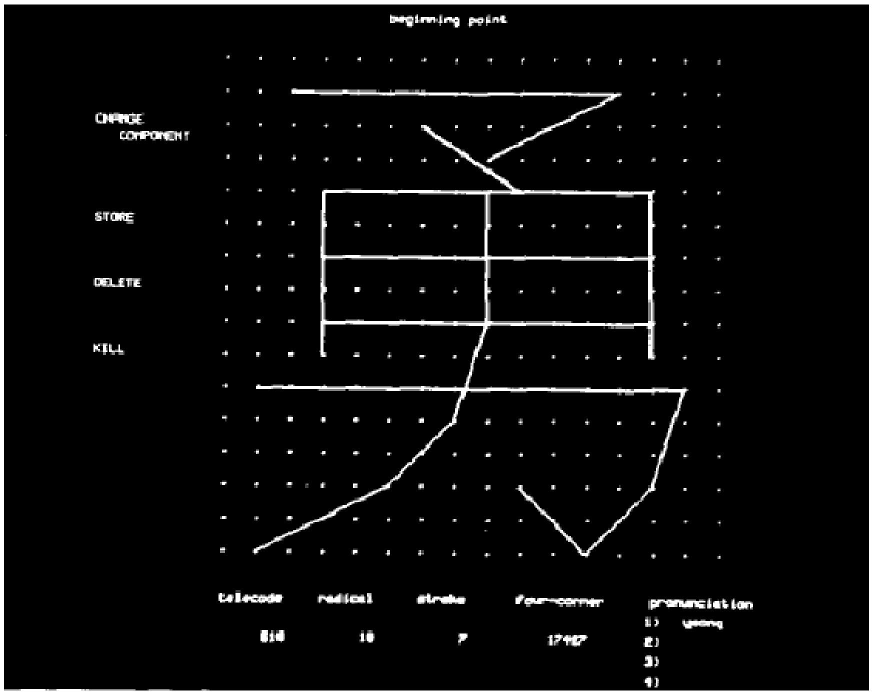


Figure 2-1 Requiring 32 Vertices, the Chinese Symbol for Brave as displayed by the 1968 Harvard Project presented by Hayashi *et al*

Over the past four decades, visual feedback has progressed significantly beyond the 16x16 grid used by Hayashi *et al.* Technologies now exist for real-time processing of solid strokes with significantly improved resolutions. Today's touch screens have resolutions of 16 dots per inch (DPI) or better while digitizers for use with electromagnetic styli have resolutions beyond 600 DPI. Speed or timing information, pressure capabilities and other features such as stylus tilt are also possible using today's Tablet PCs.

For online recognition visual feedback provides immediate clues to the user on the recognition that has occurred. With the high complexity of recognizing mathematics, the ability to provide immediate clues to the user on the assumptions made by the recognizer will improve results. For instance, immediate results of online recognition could be provided through a pop-up list of most probable matches or visual clues that illustrate segmentation and grouping of ink strokes. These visual results require that devices which are targeted have a suitable means of visual display.

Devices that support offline recognition – such as the older IBM CrossPad and the newer Logitech IO Digital Pen are not targeted in this thesis or the mathematical editing framework produced by ORCCA. Although both of these devices support offline recognition and manipulation of ink, this class of device does not provide the interactivity required to ensure math recognition is as accurate as possible.

2.6 Hardware Requirements of Targeted Devices

Processing power is no longer a key obstacle in creating a mathematical recognition engine. Current handheld technologies are fast approaching gigahertz speeds, with some Pocket PCs currently containing CPUs in excess of 600 MHz. What has been recognized as being the most critical requirement is the resolution and size of the interactive screen. This factor has been the most dominant in choosing acceptable platforms to be targeted by this thesis in creating a mathematical framework.

2.6.1 Hardware used in Benchmark Tests

Results presented in this thesis were produced using hardware as illustrated in Table 2-1.

2.6.2 Interactive Screens

While collecting data for analytical use, it was noted that the size of a handwritten equation was roughly twice that of an easily visible equation, for most writers a size 24 font on a 14” display with 1024 x 768 resolution (refer to section 8.8). In addition to needing a large screen area for input, displays that support a landscape or horizontal display will be favored as most mathematical input reads left to right first, then top to bottom. While Tablet PCs and other large displays are typically square or horizontally oriented, most mobile devices are oriented vertically which is an inherent disadvantage to applications expecting long lines of input.

Most current models of Palm Pilots support a 320 x 320 screen, where as Pocket PCs support a minimum of 320 x 480, with some newer devices capable of Quarter VGA (QVGA) or 480 x 640. QVGA devices offer the same resolution as original VGA monitors, but in approximately a quarter of size, presenting users with high resolution mobile devices. Both the Palm and Pocket PCs are oriented vertically, although Pocket PCs running Windows Mobile 2003, or newer, operating systems are capable of changing to a horizontally oriented screen layout. For development and prototype purposes, 320 x 320 resolutions appear acceptable. In discovering that individuals match a size 24pt font for handwriting input, we conclude that QVGA devices should be the minimal target once a production quality application is released to end users. This higher resolution will enable individuals to write enough mathematics to make meaningful use of their device.

Table 2-1 Hardware used for benchmark algorithms implemented in this thesis

	Pocket PC	**Tablet PC**
Operating System	Windows Mobile 2003	Windows XP Tablet PC 1.0
Processor	400 MHz Intel XScale	700 MHz Pentium III
Memory	64 RAM, 64 ROM	384 MB RAM
Display Size	320 x 480	1024 x 768
Input Resolution	1 X display resolution	10 X display resolution[1]

2.6.3 Processing Power

Ink manipulation is nontrivial task, but with next Pocket PCs and Palms soon breaking gigahertz speeds as predicted by Moore's Law[21], these platforms should provide minimal processing delays. Furthermore, advances in mobile processors have become more rapid as these devices will now evolve together: the newest Palms will use

[1] Exact hardware specifications could not be found. Instead, the general guideline presented by Microsoft of the input resolution being ten times that of the monitor.

processors once built for Pocket PCs. In creating prototypes for this thesis, the most CPU intensive calculation (determining the intersections of an ink stroke) was used for benchmarking. For details on these tests, refer to section 8.10.

In creating a benchmark, we used ink strokes with 100, 500 and 2500 data points for the Tablet PC, and 25, 50 and 250 for the Pocket PC. These sizes are representative of the sizes of individual characters, a small group of characters and a simple equation, as seen in Figure 2-2.

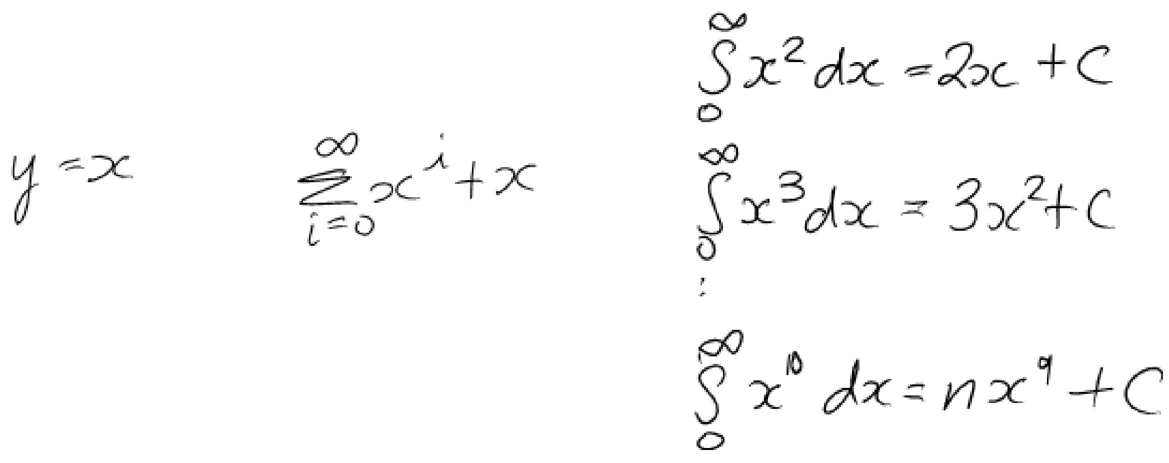

$y = x$
$\sum_{i=0}^{\infty} x^i + x$
$\int_0^\infty x^2 dx = 2x + C$
$\int_0^\infty x^3 dx = 3x^2 + C$
$\int_0^\infty x^{10} dx = nx^9 + C$

Figure 2-2 Samples of Ink from the Tablet PC, representing sets of data that include 100, 500 and 2500 data points, respectively.

Using strokes with 100, 500 and 2500 line segments took approximately 0.04, 0.28 and 1.74 seconds, respectively, to complete self intersection on the Tablet PC. At 0.28 seconds the delay in computing intersections was barely noticeable. As intersections are a possible feature that will be used in mathematical character recognition, we conclude that the Tablet PC has suitable processing power and memory requirements for math handwriting recognition.

Using the Pocket PC, times to compute self-intersections of strokes with 25, 50 and 250 line segments required .05, .25 and 1.20 seconds, respectively. Given the smaller the input region on a Pocket PC and lower resolution of the touch screens compared to a mouse or Tablet stylus, these stroke sizes are similar to Figure 2-2, although the resolution and smoothness is significantly less. None of these stroke sizes incurred noticeable delays in computing on the tested Pocket PC, implying the hardware is sufficiently powerful, given the current screen size and resolutions.

2.6.4 Memory

Memory requirements of a math recognizer are significant; expansive databases of samples will be required to ensure recognition can occur. In 1994 Jonathan Hull compiled a collection of 20,000 samples of Cities, States and ZIP codes from mail sent through the United States Postal Service [22]. Consisting of 8 bit grayscale images, the database was over 600 MB in size. Further discussed in section 8.7, I was a part of an ink data collection survey prepared by ORRCA which collected a comparable database of common math notations – approximately 20,000 samples in only 90 MB of uncompressed data. Once compressed, it is possible that only a third or less of this capacity will be needed. A 30 MB size database is a significant requirement for a handheld computer, especially as high end models currently have only 128 MB of memory. With the addition of expansion ports that support expandable memory cards, 30 MB of memory is achievable. Significant additional databases of expression samples will be needed, so we see that Pocket PCs may need to connect to remote servers for some time. We have had success with Tablet PCs and desktop computers having 512 MB of RAM, tests have shown that 394 MB is insufficient.

2.7 Targeted Software - Requirements

One of the advantages of the mathematics framework referred to in this thesis is platform independence. Furthermore, the ink architecture proposed will take advantage of supplied software support where possible, complementing missing functionality as necessary to ensure the digital ink architecture standards are implemented fully.

Although it is always possible to implement missing functionality, it remains that certain minimum software requirements must be met. Inking relies on an event supportive model; operating systems must provide the ability to capture at least three events: 1) Stylus Down, 2) Stylus Move and 3) Stylus Up.

By supporting these three minimal events, it is possible to simulate all inking actions; a stroke object simply becomes a collection of X, Y coordinates as collected by these three events. Without these events, it is impossible to create a stroke and the inking actions become impossible. We therefore define these events in more detail.

2.7.1 Stylus Down

This event is activated once the stylus is pressed against an interactive display. It must include the ability to acquire X, Y coordinates of the stylus and should also include a method of acquiring timing information. Future recognition engines may use timing information to help in recognizing ink.

2.7.2 Stylus Move

This event occurs after activating the *Stylus Down* event, while the stylus is pressed against an interactive display, movements of the stylus must be recorded in X, Y coordinates. Timing information is not necessary, as long as the *Stylus Down* and *Stylus Up* events each generate a time stamp, as data collected by *Stylus Move* could then simulate time stamps, using this information.

2.7.3 Stylus Up

This event is activated as the stylus is released from the interactive display. It is not necessary to acquire X, Y coordinates although it is necessary to collect the time of this event relative to the *Stylus Down* event.

2.8 Contrast of Targeted Platforms - Hardware

Table 2-2 illustrates a simplified overview of identified hardware features that are typical of high-end models of Tablet PCs, handhelds and graphical digital styli. The chosen hardware features represented have been identified as being the most important in a Math framework.

2.9 Contrast of Targeted Platforms - Software

Table 2-3 provides an overview of potential software platforms that can make use of a math recognition engine. It illustrates the extent of the existing infrastructure's support for inking, as well as the native purpose of the stylus for input.

Table 2-2 Comparison of Targeted Hardware Platforms

	CPU	Memory	Input Resolution	Screen Size
Tablet PC	Pentium M, 1.3+ GHz	1+ GB	1024 x 768 or equivalent, > 100 samples / second	4 – 14" or larger
Pocket PC	Intel XScale 623+ MHz	64 MB + expansion	480 x 640	3.5" – 4"
Palm Pilot	Intel XScale 400+ MHz	64 MB + expansion	320 x 320	3"
Graphical tablet with Stylus	N/A	N/A	1024 x 768 or equivalent	14" - 18"

Currently, only the Tablet PC running Windows XP Tablet edition supports a full inking architecture, providing developers a means of manipulating digital ink via the provided manufacture APIs. Tablets running on Linux as well as PDAs and graphical tablet styli typically support only mouse simulation or simpler forms of character recognition such as Palm's Graffiti recognition scheme.

Table 2-3 Comparison of Targeted Software Platforms

	Supported OS	Functionality	Primary Use
Tablet PC	Windows XP Tablet	Full ink support	Input, ink manipulation, recognition
	Linux	Limited	Input only; recognition plans for future
PDA	Windows Mobile	Basic events	Input, full character recognition
	Palm	Basic events	Input, shorthand character recognition
	Linux	Basic events	Input, shorthand character recognition
Graphical tablet with Stylus	Windows	Mouse input	Graphical applications
	Linux	Mouse input	Graphical applications

At this time a large number of targeted platforms - handheld computers as well as Tablets running Linux - do not provide a means of ink manipulation. To accommodate the lack in functionality it will be necessary to provide a Portable Digital Ink Architecture (PDIA) to ensure that full ink manipulation and API support is provided. This will allow a level of abstraction allowing a single math recognition engine to operate on all platforms.

2.10 Conclusion

In creating a parallel between the use of early tools in aiding mathematics and a mathematical handwriting recognizer, we argue for the importance of such a tool. By establishing the requirements of a recognizer both in hardware and software specifications, we ensure that our solution will provide an experience that is positive, productive and natural.

The three software requirements; stylus down, stylus move and stylus up are essential if inking actions are to be provided to users. Furthermore, the requirements of hardware, including interactive screen, processing power and memory, ensure real time interaction with the user will be possible, although complex mathematical equations may cause delays. The importance of real time interaction and feedback and its impacts on user interfaces is further discussed in Chapter 7

Chapter 3 Attributes of a Mathematical Ink Recognition Environment

3.1 Overview

To ensure the success of the mathematical recognition engine developed by ORCCA, this thesis has identified the need for a digital ink architectural framework. This chapter describes ***why*** attributes of a mathematical recognition are chosen. Chapter 4: Portable Digital Ink Architecture (PDIA) describes in detail ***how*** each attribute should be addressed in our solution. Chapter 8: Implementation, Experiments and Results explain the ***results*** of our experiences in implementing PDIA.

The requirements necessary for a mathematical recognition engine are not unrelated to a standard handwriting recognition engine; many could be addressed by increasing minimal resource requirements. Instead it is the requirements that will make a math recognition engine *successful* that differs significantly from handwriting recognition.

Handwriting recognizers, either through shorthand notation or full text recognition schemes, are inherently successful or must evolve to become successful. The release of Apple's Newton in 1993 illustrates this notion. The Newton device was well equipped and offered a wide range of software; however it lacked accurate handwriting recognition. The Newton was discontinued in 1998 after a second release of the product failed to reverse market opinions of the first generation Newton. By failing in handwriting recognition, the Newton lost its audience which moved to alternative solutions that had evolved into more successful products.

Manufactures are aware of the importance of handwriting recognition in pen-based computer systems. To consumers, handwriting recognition is not a feature of pen computing, it is a requirement. The result is proprietary technologies that target single devices offered by each manufacture, in an attempt to provide a competitive advantage and incentive to use a single device. One exception is Palm's Graffiti software which is available for licensing to other manufactures.

Further driving the need for accurate handwriting recognition is the diverse range of software applications that provide opportunity for input by ink, utilizing the capabilities of text recognition engines. These programs offer customers a suite of applications in many areas of computing and the result is a large market base. If one category of software titles were to fail as a result of poor recognition results, others might continue until accuracy improves.

While the proprietary approach has worked for handwriting recognition, it is less desirable for mathematical recognition for two reasons. First, a mathematical recognition solution should have multiple targets spanning dozens of manufactures and several platforms (including Windows XP and XP Tablet edition, Windows Mobile, Palm, Linux and others). While solutions have existed in the past that have been capable of spanning multiple platforms, they often take the "lowest common denominator" approach. Mathematical recognition will require high level ink manipulation, or the ability to manipulate ink though a comprehensive set of APIs; the result is a need to identify a "common denominator" without being bound by the "lowest denominator". Once this denominator is discovered, it will be necessary to implement missing functionalities on each platform.

Second, although individual platforms may or may not have accurate recognition capabilities, most have comprehensive list of software titles that take advantage of the existing recognition support. While some of these titles may fail because of poor recognition, others will survive until a better recognizer is made available. In the end the platform endures because of the success of its applications, something a math framework is not guaranteed.

A math recognizer will have only a limited number of software titles, all of which will be dedicated to mathematics: a virtual calculator representing previous interfaces at worst, or a whiteboard with the ability to allow free form entry solving domain independent equations at best. While useful, the market size for math recognition API is considerably smaller than a general handwriting recognition API. The result is undesirable conditions for 1) potential developers – poor adaptation of API and 2) consumers – fewer software titles to choose from.

3.2 Portable Digital Ink Architecture

The need for a PDIA and the ability to directly access or analyze ink collected within a document is more prominent in a mathematically oriented application than typical handwriting recognition applications. Most handwriting to text applications are required to do little analysis beyond the grouping of textual characters. Math applications however, need to be able to recognizer symbols beyond text, such as fractional lines, matrix borders and many others.

To ensure as successful as possible adoption of our mathematical framework and recognition engine, we identify a need to include with our solution a Portable Digital Ink Architecture. Our proposed math recognition solution will need to address the following requirements[1], made possible with PDIA. These requirements will ensure the greatest possibility of success while preventing an unfeasibly large set of features from creeping into our designs.

1) Platform Independence
2) Consistent High-Level Ink Manipulation
3) Abstract Device API Evolution
4) Abstract Resource Availability

3.3 Platform Independence

Often considered a *holy grail* for many architectures, platform independence is an essential requirement for a math recognition application. As identified in sections 2.4

[1] This section describes why each requirement has been identified. Please refer to Chapter 4 for details on how each requirement is implemented in our solution.

through to 2.9, there are several hardware devices each with different operating systems that are targeted devices for our math framework and recognition engine. While each combination of hardware and operating systems has a positive future in mathematical pen computing, it will be necessary to target as large a subset as possible to maximize the likelihood of success. Similar to Sun Microsystem's platform independent Java and Linux's hardware independent nature, the benefits of portability is in the choices presented to both developers and users. In the case of our math framework, it will be possible for third party developers to take advantage of the unified inking interface and math recognition engine in a manner that addresses their specific needs. For users, it means choices in finding software that runs on devices they prefer to use.

Platform independence is closely related to the requirement identified in section 3.4, High-Level Ink Manipulation. Platform independence when combined with inking capabilities offers the potential for a single recognition engine to operate consistently on each targeted platform as described in the following section.

3.4 Consistent High-Level Ink Manipulation

The ability to treat *Ink* as a native data object is not unique, yet many platforms including all current Palm and Pocket PC handhelds do not come with manufacturer support for ink manipulation. The creation of a mathematical recognition engine and an appropriate user interface will require full support of ink manipulation, well beyond the limited X, Y coordinate reference support currently provided by handheld manufactures.

Providing a uniform API across each of the targeted platforms will be necessary to ensure a single recognition engine supports each platform, providing a consistent means of easily manipulating ink. Missing functionalities from the manufacturer APIs will need to be implemented, yet it will also be desirable to take advantage of existing functionalities and data structures, as in the case of the Tablet PC.

In addition to consistency between devices, a homogeneous API provides an opportunity to address the next requirement; that PDIA easily accommodates device and API evolution.

3.5 Device Evolution Abstraction

Given the fast pace of change in the mobility market, it is reasonable to anticipate that device APIs will not remain constant. Two examples of this notion are[1]:

1) Palm's 2002 release of the Palm OS 5[23]; since their 1996 debut Palm has averaged 19 months per release as of 2004.
2) Microsoft's upcoming Tablet PC version 1.7[24] due to be released in 2005 is the third revision to the Tablet PC API since 2000, is also averaging 19 months per version.

Further evidence suggests that the Tablet or Pocket APIs may change again within the next year; internally Microsoft moved the Tablet PC to the Windows Mobile team and announced that a new generation of small 5 inch Tablet PCs will ship in 2005[25]. It is foreseeable that these new devices will merge with the Pocket PC platform, or potentially acquire a lightweight implementation of the Tablet API.

As the use of a standardized ink interface will require the addition of an abstraction level on top of manufacture inking support, this abstraction could also be used to accommodate rapidly changing APIs. Additionally, this layer of abstraction can also be used to accommodate the remaining requirement – that algorithms and calculations be aware of the platform of execution, ensuring timely results and appropriate CPU usage respectively.

[1] Palm's period of release was calculated from their 1996 debut though to 2004, using their latest OS release. Information regarding betas were not included. Microsoft's Tablet release period was determined using beta release dates, originating in 2000 through to the expected 2005 release of version 1.7

3.6 Abstract Resource Availability

It is acceptable to assume significant amounts of resources are available to perform recognition on a Tablet PC, or on a remote server which is also expected to exceed minimal resource requirements. Moving to a handheld platform, however, could result in code that is bulky and use more than the available amounts of memory and CPU processing power.

It will be necessary for PDIA to be aware of each platform's limitations and to provide implementations of algorithms that are sensitive to these restrictions. It may be that approximate algorithms are implemented, or algorithms that are aware of the type and amount of input expected. For instance, a device with a high resolution electromagnetic screen versus a touch screen will provide much more data, often significantly more than necessary. The touch screen device however, will provide only a minimal amount of input data, all of which is likely to be significant. By being aware of the platforms hardware and the expected amounts of input, it is possible to influence initial object sizes in memory, as well as determine thresholds that influence when one algorithm should be used over another. For example, devices with large amounts of input will use an intersection detection algorithm that has a high overhead but low average runtime complexity (n log(n)), with n representing the number of data points collected. Devices with low amounts of input can use an intersection algorithm with no overhead but a higher runtime (polynomial).

3.7 Conclusions

Each of these four requirements: Platform independence, High-level ink manipulation, Device API abstraction and Resource abstraction can be addressed in a well-crafted implementation of PDIA. The next chapter illustrates how precisely defined interfaces and abstraction layers provide a means to address each requirement in a reasonably lightweight manner.

Chapter 4 Portable Digital Ink Architecture

4.1 Overview of Portable Digital Ink Architecture (PDIA)

As indicated in [26], the domain of mobile computing is diverging, just as desktop computing was two decades ago. As in early days of home computing when computer manufactures failed to recognize the benefits of standards, often while producing incompatible products within their own range of devices, similar situations are seen today. The mobile computing market has numerous examples of devices that exclude alternative products. While standards have ensured that cellular phones are generally capable of broadcasting and receiving calls from different manufacturers and networks, there are no standards in place to permit these same phones from sending contact or calendar information to each other.

This thesis introduces a common architecture centered on several device types, each of which is manufactured by a separate company and whose primary purpose varies. This chapter is dedicated to overcoming the specific shortcomings of handheld, tablet and desktop platforms, providing a means to implement core application logic only once for execution on each targeted platform.

Section 4.2 introduces the three-tier architectural model created for this thesis and identifies how each of the attributes of a math API (platform independence, ink manipulation, API evolution and resource limitations) will be addressed in PDIA. Section 4.3 identifies the objects necessary for ink manipulation and Sections 4.4 and 4.5 present problems in persistent storage of ink (existing ink formats) and identifies our decision to use the standardized InkML [27] to represent ink.

4.2 *Creating an Architecture for PDIA.*

Presented in this section is an examination of how the proposed architecture from that of manufacturer approaches to the three-tier architecture that is capable of platform independence. Using the modular approach presented also allows for the development of additional functionality by both internal and external developers working with PDIA, providing users with the option to use third party applications.

4.2.1 Manufacturer Approach: Single Tier Architectures

Manufacturers that currently support Ink, notably Microsoft with the Tablet PC, provide an uncompromising implementation of their Architecture. The single tier architecture, illustrated in Figure 4-3, clearly benefits manufactures by constraining developers and consumers to a single device or platform.

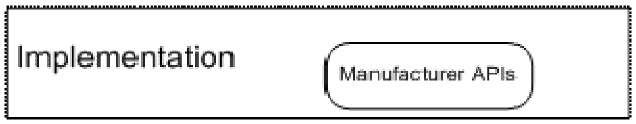


Figure 4-3 Single Tier Architecture: Manufacturer Implementations

This single stage architecture provides only an *implementation* of inking APIs. In this single tier architecture resides all responsibility for Ink manipulation, including the ability to modify, recognize or serialize ink to a persistent storage medium. These capabilities are provided through private or publicly-provided manufacturer APIs and are often a part of the operating system. With little or no flexibility in these platforms, no single manufacture architecture is suitable for use in PDIA. We therefore introduce a second tier into our architecture, providing a manufacturer and platform-independent architecture.

4.2.2 Platform Independence: Two Tier Architectures

When targeting multiple devices, platforms often introduce a wrapper that achieves portability by supporting a lowest common denominator approach. With a math API and our targeted devices, the lowest common denominator result would be that of handheld computers and the ability to draw a point at a given X, Y coordinate. Requiring the full support of ink manipulation, our math framework will have to approach platform independence with a more feature rich solution.

The introduction of a second tier in our architecture provides the ability to give each targeted device any features necessary for ink manipulation. With this new layer come changes to the foundational *implementation* tier, as illustrated in Figure 4-4. As all devices are expected to provide complete support for ink manipulation, it is necessary to produce a custom implementation. This permits for device wrappers that simply make use of manufacturer APIs or when necessary a hybrid combination of our custom and manufacturer APIs.

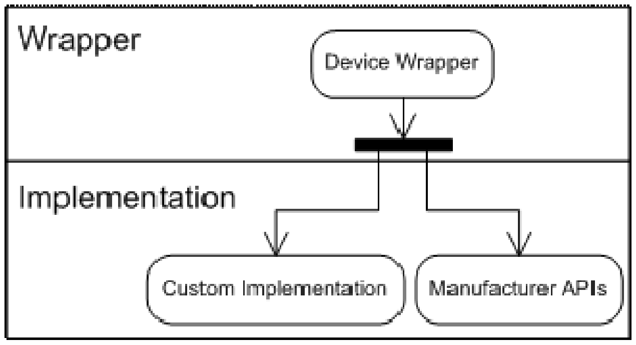


Figure 4-4 Two Tier Architecture: Abstracting Manufacture APIs

Introducing a "*device wrapper*" tier to our architecture provides the agility necessary to accommodate multiple devices and operating systems. Wrappers are meant to be the only method code that interacts with native APIs; they should be as thin as possible, incurring a minimal penalty in resource overhead. The result is a concentrated area of source code that will need to be updated should native APIs be modified. Similarly, if a new device or platform were to be targeted by PDIA, the addition of a new wrapper would permit easy expansion onto that device.

The creation of a wrapper for each platform targeted by PDIA presents an opportunity to take advantage of existing infrastructure. These wrappers also provide the ability to optimize functionality for the expected type and amount of data received while observing resource limitations. For example, wrappers on handhelds can expect much coarser stroke information with X, Y coordinates in integers. Tablet PCs, on the other hand, will require floating point numbers to record X, Y coordinate information.

Third party developers who make use of PDIA will want to interact with these wrappers, and even create new wrappers, expanding the range of devices that can be used. With consistency being a key requirement of a math framework, we wish to enforce standardization between device wrappers. We therefore need to introduce a third tier to our architectural design.

4.2.3 Standardization and PDIA: Three Tier Architecture

By permitting the addition of new wrappers to any device, the need for consistency is essential if PDIA is to meet the needs of a math framework. To enforce consistency between device wrappers we introduce a third tier to our design, a *unified interface*.

The *unified interface* layer provides consumers of PDIA with the only public interface necessary to manipulate ink for recognition or other purposes. The public methods of this layer are the product of this thesis, and may be the primary means of ink manipulation for forthcoming ORCCA projects, including a math recognition engine[1]. Each interface defined within the *Portable Interface* is guaranteed to be implemented fully in its respective device wrapper, using custom or native APIs. The necessity of implementation and conformity guarantees that PDIA yields consistent and reliable results across platform borders.

Figure 4-5 illustrates the three tiers of our PDIA solution as implemented by this thesis. In our implementation, we also provide an extension of PDIA to the Microsoft .Net framework. We leave it to future persons to extend PDIA to Java through Java Native Interfaces (JNI). Extending PDIA to .Net will enable developers to create user interfaces that take advantage of rapid prototyping applications while providing the full power of PDIA.

[1] Presented by Dr. Stephen Watt and Xiaofang Xie at ECCAD, May 20004.

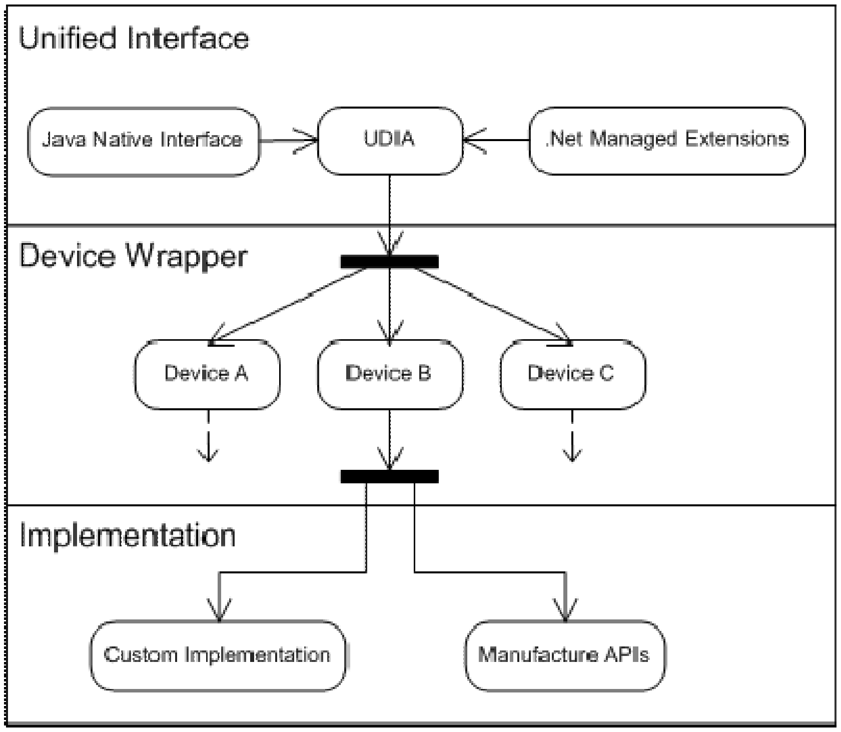


Figure 4-5 Three Tier Architecture: The PDIA Solution

Together the *implementation*, *device wrapper* and *unified interface* produce a three-tier architecture which fulfills the identified requirements for unified digital ink architecture, which were first identified in Chapter 3. The *unified interface* ensures consistency between device and platform implementations by providing guidelines for developers to adhere to. The *device wrapper* allows for cross device and cross platform support, enabling the abstracting away of device API evolution and resource abstraction. Finally by providing a complete, custom *implementation* we provide the necessary functionalities to ensure a transparent inking experience can be achieved on each targeted platform, completing an architecture that is fully independent from other operating systems, platforms and devices.

4.3 Class Hierarchical Design of PDIA

Similar to the PEN architecture described in [28], we recognize the need to "generalize internal structures beyond the hardware used" [28]. In Xiaojie's Wu's work, "Achieving Interoperability of Pen Computing among Heterogeneous Devices and Digital Ink

Formats" [29], she presents the similarities and differences between the Tablet PC and IBM CrossPad APIs. It was Wu's paper that introduced the concept of an *Ink Abstraction API*. Figure 4-6 illustrates the classes Wu identified as to allow for the greatest common functionality between the two devices of focus in her thesis, the Tablet PC and CrossPad.

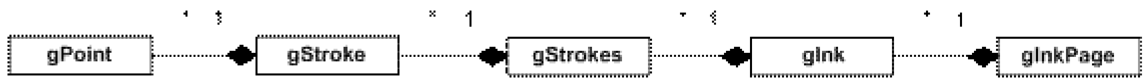


Figure 4-6 Identified Classes by Wu in an Abstraction API

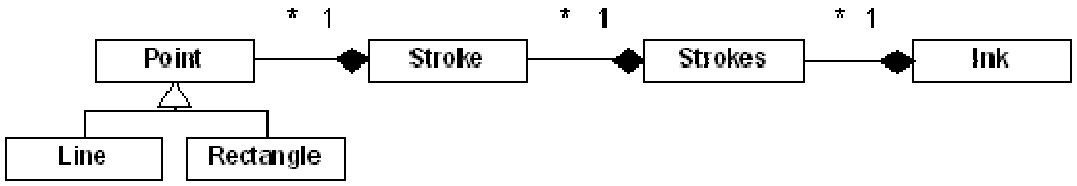


Figure 4-7 PDIA Class Hierarchical Overview

To accommodate as many platforms and devices as possible, we continue Wu's work in creating the abstraction layers of PDIA. This thesis extends her proposal, through an architecture that operates on each of the platforms targeted by a math recognition engine.

In creating PDIA we realize that ink classes, as well as those necessary to support inking, need to be made publicly available. The relationships between classes of our custom implementation of the inking inexperience are illustrated in Figure 4-7. The result was a Point class with three composite classes: Stroke, Strokes, and Ink. There is also a need for two child classes of Point: Line and Rectangle. We believe our solution is considered to be the lowest denominator possible while still supporting ink manipulation and recognition. In fulfilling the need to supplement existing APIs with missing functionality, it was necessary to provide a complete implementation of these classes, which we have made available in the namespace "OpenInk". For additional details on implementation and experiment results, please refer to Chapter 8.

4.3.1 Class Objects: Point

The Point class represents a two-dimensional vertex that contains X and Y coordinates and definitions deemed necessary of a Point object. The unit of measure for the coordinates should be flexible; most handhelds could use an integer whereas a Tablet PC may require a decimal object. It is suggested that a typedef of the unit desired is used, which can be instantiated in a configuration file unique to each device. As most languages read from the upper left to lower right, it was also appropriate to use the fourth quadrant representation as being positive for the Point object. Figure 4-8 illustrates how data in the fourth quadrant is represented, as well as the remaining three quadrants.

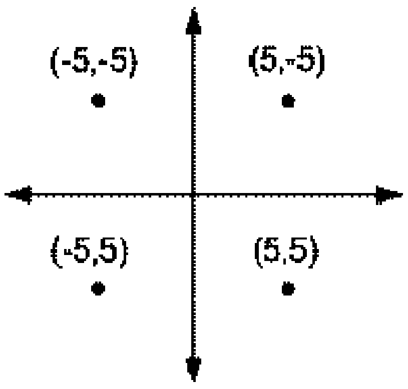


Figure 4-8 Example of syntax used to describe *Points* in each quadrant

4.3.2 Class Objects: Line

While a Stroke object is conceptualized as a series of Line segments, it is in fact a collection of Points. Not required for inking, the Line object, illustrated in Figure 4-9, allows developers to visualize functionality such as intersections within other Line or Rectangle objects. Composed of two Point objects, a source and target, the primary purpose of the Line class will remain the aiding of low-level ink manipulations.

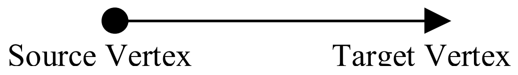


Figure 4-9 Illustration of the components of a *Line* object

4.3.3 Class Objects: Rectangle

The Rectangle is also composed of two Points: upper left and lower right as seen in Figure 4-10. The primary use of a Rectangle is to easily identify the bounding box of Strokes objects. Other examples of use include: Scaling a Stroke to a particular Rectangle, Stroke normalization by removing Points beyond a specified Rectangle or for hit tests that determine if the bounding boxes of two Strokes overlap each other.

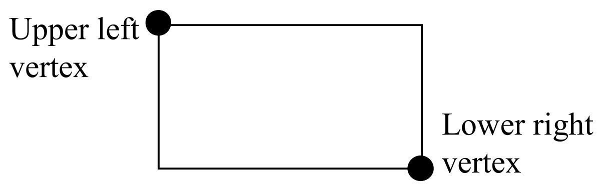


Figure 4-10 Illustration of the components of a *Rectangle* object

4.3.4 Class Objects: Stroke

A Stroke consists of a continuous set of Points representing a single pen stroke, normally supplied by an end user but may be added programmatically. The Stroke is the heart of the inking interface, and is a means of treating ink as a native data type instead of an image. Because all ink recognition and manipulation must use data located within the Stroke object, it is necessary to provide a sufficient set of accessing and manipulative functions. Examples of such functionality include the ability to find bounding boxes and intersections, stroke normalization, rotation and skewing. As the stroke is a primitive object in the digital ink framework, all functions must run in optimal time and space complexities.

4.3.5 Class Objects: Strokes

Conceptually, a Strokes object could be visualized as an organized grouping of Stroke objects, forming a distinguished object. For instance, Figure 4-11illustrates how three Stroke objects are needed to create a Strokes representation of the letter ‘I’. Furthermore, Strokes objects can be combined to create larger sets of strokes objects, as illustrated in Figure 4-12. Here the Strokes objects are the results of combining two smaller Strokes, each of which contained a word: “hello” and “world”.

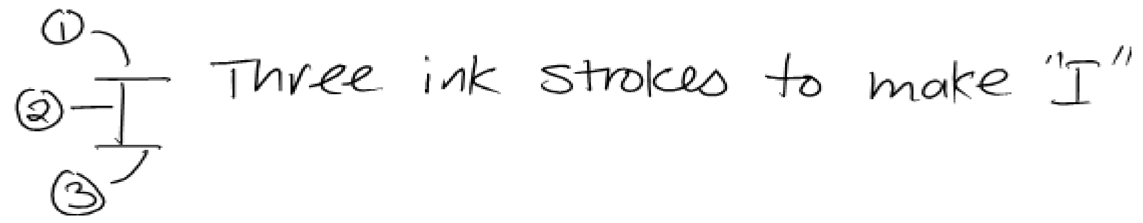


Figure 4-11 Illustrating the components of a *Strokes* Object

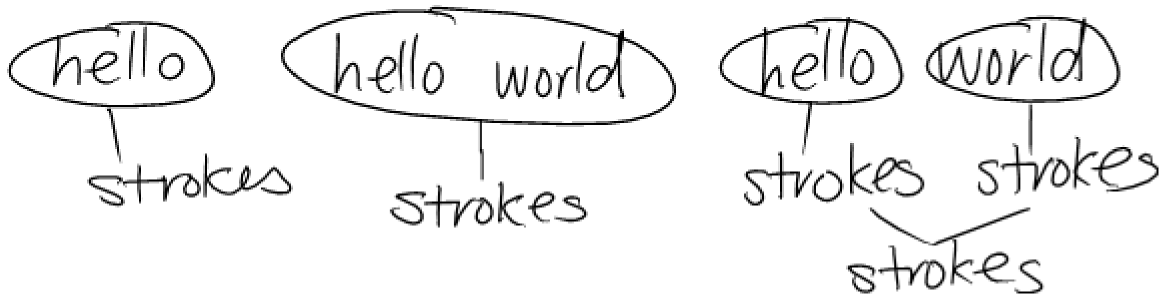
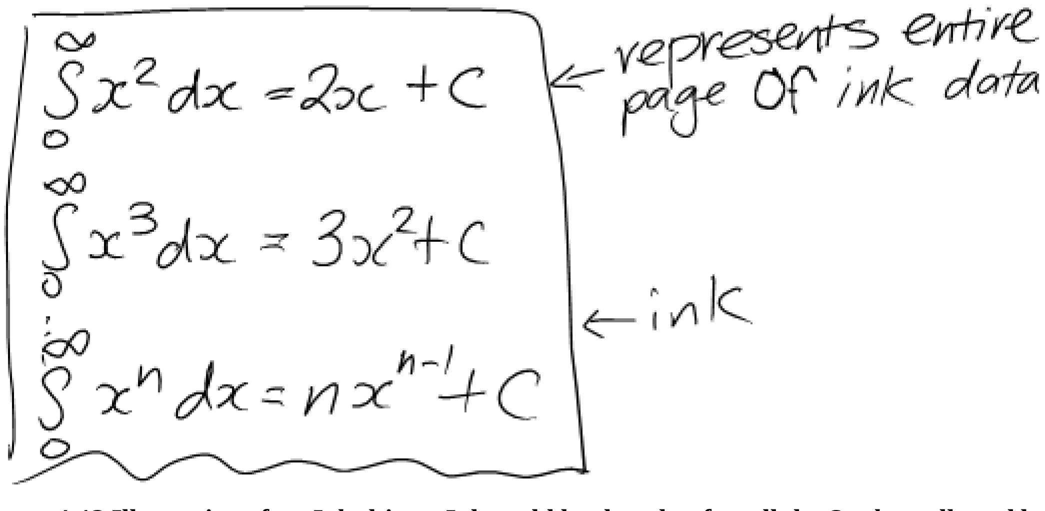


Figure 4-12 Additional Illustration of *Strokes* Objects



$$\int_0^\infty x^2\,dx = 2x + C$$

$$\int_0^\infty x^3\,dx = 3x^2 + C$$

$$\int_0^\infty x^n\,dx = nx^{n-1} + C$$

Figure 4-13 Illustration of an *Ink* object. Ink could be thought of as all the *Strokes* collected by a logical container, such as a page or screen.

4.3.6 Class Objects: Ink

Ink is the hierarchical manager of all Ink recorded from an end user, and is the primary interface for external applications. Strokes are added and destroyed through Ink, and it is the Ink object's responsibility to provide serialization capabilities for persistent storage of Ink data. Figure 4-13 gives an example of where Ink objects reside, in context to other Strokes, Stroke and the other data types presented.

4.4 Persistent Storage of Ink

Further paralleling the problems of early personal computers with today's mobile devices are inconsistencies in communicating between applications on devices. Today the possibility of cross-platform data exchange between mobile devices is moderate at best. Many devices are not even capable of communication with others; cell phones for example use proprietary methods to store contact or calendar information or fail to provide the hardware necessary for exchanging this data. Magerkurth states in [26] that "rarely is there a way to exchange this simple data even if similar communication hardware and protocols are used", referring to the problems of exchanging data between mobile devices.

With contact and calendar exchange formats, a single standard has been imposed by the Internet Mail Consortium[1] for nearly a decade. Unfortunately many cellular device manufactures continue to ignore this standard today. Given the high degree of complexity of inking, compared to calendar or text information, along with the lethargic embracing of digital ink by industry, it is unfortunate that adaptation of standards continue to be a problem. The results are proprietary and device-dependent ink formats that have become the topic of this thesis and others, alike [29, 26].

[1] vCard (.vcf) an electronic business card and vCalander (.vcs) an electronic calendaring and scheduling exchange format are trademarks of Internet Mail Consortium. Introduced in 1996, these two standards were developed to ensure communication between electronic devices is quick, reliable stored, organized and easily located. For full details, visit http://www.imc.org/pdi

4.5 *Existing Formats of Ink*

With the establishment of the internal memory representation of ink in section 4.3, it is necessary to identify how Ink will be saved to a storage medium. Given that three major existing ink formats exist (Jot, UNIPEN and InkXML), it is logical to implement one of these formats as opposed to creating another format specific to PDIA. Drawing on Wu's conclusions from her research in creating interoperability between digital ink formats [29], we present a summary of each ink format:

4.5.1 Jot

A now defunct ink format, Jot was a proprietary format owned originally by a Slate Corporation. Jot offered limited opportunities as a communication medium. A light weight binary format, it was based on a lossless compression scheme and included abilities to reduce the amount of information retained in ink.

Supported ink features of Jot include: multiple strokes of ink, bounds, scale, offset, color with opacity, pen tips, timing information, height of the pen over the digitizer, stylus pressure, button usage of stylus and, X, Y tilt angles of stylus. Applications can choose to recognize or ignore properties as required. Because existing applications may choose to ignore properties, new features could be added without affecting their performance.

4.5.2 UNIPEN

Designed in 1993 by a consortium of over 40 companies, UNIPEN was created to facilitate digital ink data exchange and the storage of handwriting samples. Using a flat attribute organization in ASCII format, it uses self-definition from three basic keywords: *comment*, *reserve* and *keyword*.

Through self definition, UNIPEN is able to record all attributes of ink. New attributes may be added as desired, as applications are hard-coded to recognize only those of interest.

4.5.3 InkML

An open standard InkML[1] was designed to replace the flat attributes of UNIPEN with an XML based schema. Promoting the exchange of data across heterogeneous devices, InkML is poised to become the de-facto standard if manufacturers adopt it.

With the option for binary encoding, InkML's usage is divided into three main categories: Ink streaming applications (instant messaging, whiteboards), persistent storage, and interactive ink (gestures). Furthermore, each XML element has both a primitive, standardized element as well as the ability to add application specific information. This allows for additional information to be added without the need to update existing applications.

Due to the standardized nature of XML, PDIA will embrace the InkML working draft standards as our choice of a digital ink format. InkML provides the greatest flexibility for PDIA, and meets both our current and future needs: Currently, the ability to serialize ink, storing it on persistent storage is necessary, however as addressed in Section 9.3, future additions to PDIA may include networkability. Designed with streaming abilities, InkML is an obvious choice to ensure these future additions are possible.

4.6 Conclusions on PDIA

Commercially, investment in digital ink is linked to risk and isolated market opportunities. Unlike desktop computing, there exists no dominant platform. Those who use digital ink want both ultra mobility (PDA's) as well as processing power (Tablets and laptops). These hardware alternatives, when combined with a selection of operating systems result in further proprietary development models, inconsistent APIs, ink formats and fragmented adoption opportunities.

Our PDIA design hopes to remove these risks, while making a mathematical framework possible. Taking advantage of existing infrastructure, PDIA ensures as little overhead as

[1] In 2002 the World Wide Web Consortium started to develop a standardized ink format *InkML* (http://www.w3.org/TR/InkML), but industry has not yet widely adapted this unfinished standard.

possible while meeting all of the requirements of a math framework: Platform independence, consistent ink manipulation, device evolution abstraction and resource abstraction.

Our three-tier approach ensures functionality is optimized for the type of data received and that resource limitations are observed. Initial versions of PDIA will support the desktop, Tablet PC and Pocket PC platforms. As expected, thin wrappers will sit on top of manufacturer implementations, with PDIA implementing missing functionality.

Chapter 5 Certain Mathematical Expression Properties

5.1 Overview

In order to understand the full relevance and difficulties of creating a math recognizer or framework, it is necessary to understand some of the properties of mathematical expressions. After examining these properties it will be possible to provide further insight as to how a mathematical framework can assist the recognition process, further contributing to our thesis. Presented in this chapter is an overview of the properties and common notations found within mathematical expressions. After examining the properties of mathematical expressions, we introduce the differences of mathematics or *two dimensional* languages with those of text based or *one dimensional* languages.

5.2 Properties and Notations of Mathematical Expressions

Thirty years ago, W. Martin suggested that the first step in automating mathematical recognition is to ensure the notation is well defined and studied [30]. Further supporting his hypothesis and the need for standards-like consistency in mathematics, Martin presented a list of ambiguous conventions used by mathematicians in technical publications at the time of publication. Table 5-4 illustrates some of the expressions Martin presented.

We also believe that before one can recognize a mathematical expression, it is important to understand how such an expression is constituted. Unfortunately, mathematics includes both hard (well-defined) and soft (poorly or undefined) conventions [31]. Examples of hard notations include the meaning of an expression, its characters and symbols: i.e., $\sum$ (***sigma***) is understood to represent the sum of ***a series*** of terms. A soft convention is often illustrated by the position of information around a given operator. In Table 5-5 two pairs of equations are presented that are generally understood to have the same conceptual meaning. Soft conventions permit the placement of informational variables which interact with the primary operators in different locations.

Table 5-4 Examples of Ambiguous forms of equations with their Unambiguous possible definitions, as presented by Martin

Ambiguous	Unambiguous
$\begin{matrix} a \\ \overline{} \\ b \\ \overline{} \\ c \end{matrix}$	$\frac{\frac{a}{b}}{c}$ or $\frac{a}{\frac{b}{c}}$
$\begin{matrix} 100 \\ \sum i \\ 10 \\ \sum j \\ 5 \end{matrix}$	$\sum_{5}^{\sum_{10}^{100} i} j$ or $\sum_{\sum_{5}^{10} j}^{100} i$
$\int_a^b \frac{-c+d}{2}\, dx$	$\int_a^b \frac{-c+d}{2} dx$ or $\int_a^{b-c} \frac{+d}{2} dx$
$\frac{x \frac{y}{} + 2 + w}{2}$	$x^y - \frac{+2+w}{2}$ or $x^{y+2} - \frac{+w}{2}$
$\sum_{i=0}^{\frac{a}{c+d+100}} i^2$	$\sum_{i=0}^{a/(c+d+100)} i^2$ or $\frac{a}{\left(\sum_{i=0}^{c+d+100} i^2\right)}$
x^y	x^y or the pre-subscript ${}_x y$
$A(B)$ $A2$ $A \times B/C$	$A * B$ or the function $A(B)$ $A * 2$ or the variable $A2$ $\frac{A*B}{C}$, $\frac{A \times B}{C}$ (variable x) or $A * \frac{B}{C}$
$\sum_{i=5}^{10} i + Y$	$\left(\sum_{i=5}^{10} i\right) + Y$ or $\sum_{i=5}^{10}(i+Y)$
$\frac{dx}{dt}$	$\frac{d*x}{d*t}$ or $\frac{d}{dt}x$

In both the integral and summation formulas of Table 5-5, it is acceptable if these two minor variations in variable placement are used interchangeably. To a math recognizer looking for operator parameters, such ambiguity in parameter placement will require additional CPU processing, or additional user interactions, further complicating the recognition process.

Table 5-5 Two pairs of equations that are understood by hard conventions to have the same meaning. Both pairs use soft conventions to place the primary operator limits in different positions

$\displaystyle\int\limits_0^{2\pi} 2x\,dx$	$\int_0^{2\pi} 2x\,dx$
$\displaystyle\sum_{i=0}^{10} x^i$	$\sum_{i=0}^{10} x^i$

To prevent undertaking a complete survey of the notation of mathematics, we limit ourselves to certain factors that influence recognition, as listed below. Believed to have significant impacts on expression meanings, we further discuss each factor and related sub-factors in upcoming sections.

1) Symbol Identification: Factors include implicit and explicit operators and identify characters versus operators.
2) Segmentation: Factors include operator precedence and range, use of fence and binding operators and operator symbols that imply grouping.
3) Context: Related to categories 1 and 2, it is necessary to ensure a symbol has been correctly recognized according to its context, allowing proper grouping and identification to occur.

5.2.1 Symbol Identification

Symbol identification involves identifying symbols that will result in implicit or explicit operators, as well as identifying operands versus operators.

5.2.1.1 Symbol Identification – Operators and Operands

The correct identity of a symbol influences every step of the recognition process. Often it is possible to determine if a character is an operator or operand, which helps determine if an implicit or explicit operator is implied. One significant problem with identifying a symbol in mathematics instead of a written language was identified by Blostein *et al* [32]:

> … there is a large character set (roman letters, greek letters, operator symbols) with a variety of typefaces (normal, bold, italic), and a range of font sizes (subscripts, superscripts, limit expressions). Certain symbols have an enormous range of possible scales (e.g. brackets, parentheses, $\sum$, $\prod$, $\int$).

5.2.1.2 Symbol Identification – Explicit or Implicit Operators

As said, symbols may either be operators or operands, although there are exceptions to this as well. Usually, if a symbol is defined as an operator, it is an explicit operator, but this is not always true. More difficult to recognize are implicit operands, because of their subjective nature. Chan *et al* [33] provides the following definition and excellent example on implicit operators:

> Also called spatial operators, relationships between operands are identified by implicit operators by their relative position. For example, in "a^2", ***2*** is the superscript of ***a*** representing the square of ***a***. However, in "a_2" ***2*** is the subscript of ***a*** representing only a variable name. Although unusual, "***a2***" can be used to represent the multiplication of ***a*** and ***2***.

5.2.2 Segmentation

In identifying segmentation properties of mathematical expressions, it is necessary to examine operator precedence and range, the use of fence and binding operators as well as operator symbols that imply grouping.

5.2.2.1 Segmentation – Basic and Combined Symbols

Even if each character has a well defined meaning, it is possible that symbols grouped together will have another meaning. When formed together, digits typically become a unit: i.e., ***22*** represents an integer, twenty two; however 2^2 represents a different integer, four. Likewise characters may be grouped together to form units. Examples of common units of characters include ***sin***, ***cos***, and ***tan***. To decide if individual symbols are to be grouped, it is the context of the characters that will provide this information.

5.2.2.2 Segmentation – Range Operators

Each operator has a pre-determined *range* over which it is effective, which is typically based on conventional usage. The example in Table 5-5 illustrates how these ranges differ between conventions – the bounds on the integral and summation operator appear in two different locations: Above and below the operator as well as to the left of the integral. Because mathematical notation is a *visual language* (refer to section 5.3), the use of subtle spatial relationships are subject to variation. For additional examples representing ambiguity arising from subtleties in spatial relationships, refer to Table 5-4 or [30, page 83].

5.2.2.3 Segmentation – Binding Operators

Binding operators are those which create a group of symbols meant to be treated as a single unit. For instance, $\sqrt[3]{x^3 + 27}$ is meant to be a single unit bound by the root operator.

5.2.2.4 Segmentation – Fencing Operators or Precedence

Fencing operators are related to binding operators in that they group series' of operators and operands, allowing the creation of a single unit. Fencing operators are typically parentheses, but are also used in matrixes. For example in "$a(b+c)$", the parentheses groups $(b+c)$ into a single unit with a higher precedence than the implicit multiplication operator between a and $(b+c)$. As a result the sum of $(b+c)$ is calculated and then multiplied by a.

5.2.3 Context

Introduced earlier, the allowance for identical symbols to have different meanings based on context often prevents, or makes difficult, expression analysis beyond the character level. While many symbols have well defined meanings across all domains ("3" always means three, "=" always means equal) this is not true for all symbols. Fortunately, many symbols do have a well defined meaning once its context is determined. Three popular examples of symbols that depend on context include:

1) "·" This dot may mean a decimal place, multiplication operator or a symbol annotation, i.e., a time derivative or repeating digit.
2) "—" The horizontal line may mean subtraction operator or a fraction line, depending on the length and location of operators surrounding the symbol
3) "*dx*" The group of characters *dx* may have separate meanings depending on context. Clearly in $\int_0^{2\pi} 2x\,dx$, *dx* is a part of the integral expression. However in "*cy* + *dx*", it likely represents the multiplication of the symbols *d* and *x*.

These three notational categories of symbol identification, segmentation and context awareness provide a means of identifying properties of mathematical expressions. Furthermore, it is possible to differentiate natural language and mathematical notations by the differences in their respective uses of dimensions to portray additional information.

5.3 Visual versus Written Languages

Written string languages, such as most of the world's languages, are composed of characters that make use of a one dimensional direction, often left to right, right to left or top to bottom. The only meaning included with syntax is what is explicitly portrayed in the text while making use of simple notations, i.e.: spaces between words provide a means of association; special characters provide a means to emphasize, insert pauses and denote questions. These meanings are defined by each language, the rules of which are grammar.

Unlike string languages, mathematics, music and some engineering disciplines make use of the syntax of visual languages to portray a user's intentions [34]. According to Anderson [35], the primary differences between string and visual languages is the use of appropriate syntax in a visual language. Visual language syntax takes advantage of higher dimensions in portraying the intended meaning. Marriott *et al* [34] more accurately defines a visual language in suggesting it is a set of diagrams whose spatial relationships are considered meaningful in the language definition.

5.4 Conclusions

In identifying the differences between mathematical or visual languages and string based or textual languages it is hoped that the recognition process can be adapted and further specialized for mathematics. Furthermore, three unique properties of mathematics that effect recognition, including: symbol identification, segmentation and context, present unique challenges that must be addressed by a math recognizer in order to provide a functional mathematical recognizer, the foundation of any end-to-end solution that will be adapted by end users.

Chapter 6 Mathematical Recognition Survey

6.1 Overview

Mathematical handwriting recognition is an area of increasing interest for ORCCA; the benefits of merging the ease of natural language input with a mathematical computation engine would be far-reaching within industry and the everyday lives of anyone involved with mathematics. This chapter presents a review of certain problems that are understood in the field of mathematical handwriting.

Introduced in this chapter is a three step process that is commonly used when attempting to recognize mathematics; then we discuss a fourth step borrowed from handwriting recognizers of natural, string based languages. In addition to the popularly used three step recognition process, this borrowed idea from the recognition of *string* or natural languages is the ability to include dictionaries and grammar rulebooks during the recognition processes for additional feedback. It is believed that the addition of a dictionary or grammar book could be used to further refine and improve accuracy in future works. This is the subject of ongoing work of other members of our research group.

6.2 Overview of Recognition Process

This chapter examines the generally accepted process involved in recognizing mathematics. By summarizing research completed by other references in the field, it is hoped to demonstrate how our mathematical framework and specifically, PDIA can assist in math recognition by providing foundational layer APIs for ink manipulation. It is also recognized that a math engine would be a separate process from PDIA, as seen in Figure 6-14; PDIA would not be responsible for the implementation of such an engine.

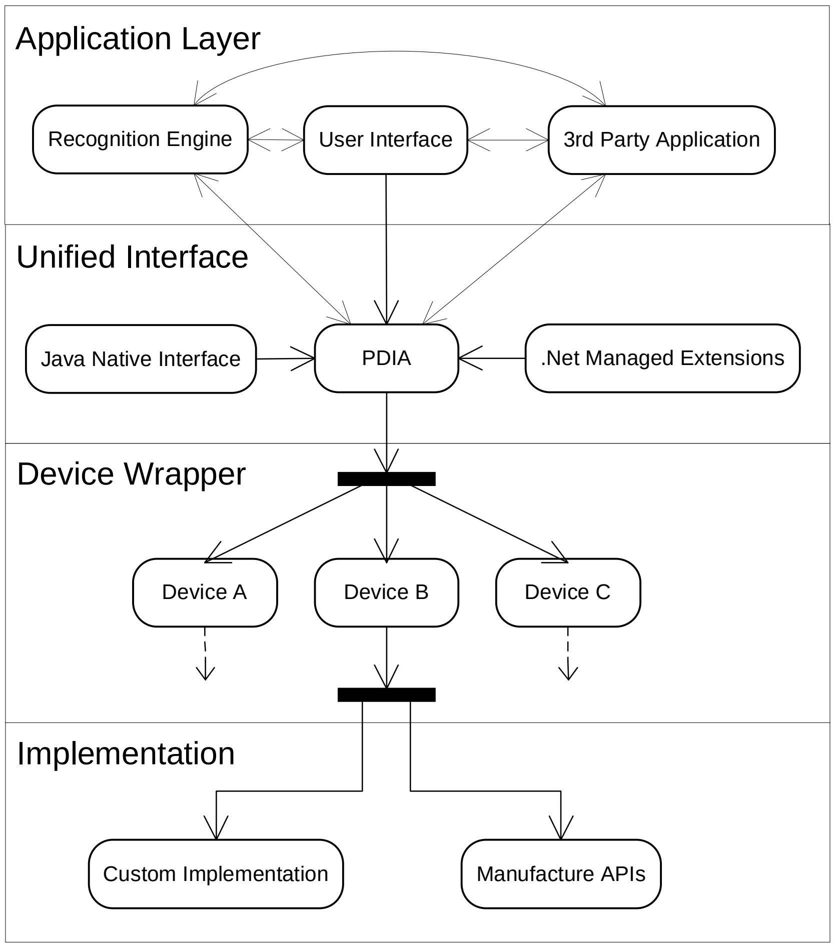


Figure 6-14 Application layer exists on top of PDIA, and may access C++, JNI or .Net extensions of PDIA

Different views or implementations of the mathematical handwriting recognition process have led to the creation of numerous stages and sub-stages in the recognition process. In upcoming sections we examine each of these stages, presenting the work that occurs along with examining how our mathematical framework and PDIA can assist the recognition process. Presented here are generalizations of the three stages that accurately represent the accepted stages in mathematical handwriting recognition:

1) Data Collection and Normalization
2) Symbol Recognition
3) Structural Analysis

A fourth step envisioned by ORCCA is also examined along with these three commonly accepted stages of recognition. Accepted as a common practice in string language recognition, post analysis of handwriting samples typically include a comparison against a dictionary or grammar rulebook. This post analysis allows for higher accuracy levels by providing feedback between a recognizer and an independent entity, the dictionary or grammar rulebook. It is our hope to capitalize on a similar process, by providing additional, automatic feedback to the recognizer in the same manner. We therefore introduce a fourth stage, which in our model examines the context of the recognized structure and compares against a "dictionary" of known mathematical expressions:

4) Context Analysis

For a more complete survey of online symbol recognition, Chan *et al.* [33] provides a comprehensive summary of several methods of symbol recognition as presented by industry peers. Methods that are adoptable to ink handwriting recognition are presented in Table 6-6.

Table 6-6 Categorization of symbol recognition methods used in different systems by Chan *et al.*

Major Method	Example
Structural feature extraction and decision tree	Beláid and Haton [36]
Flexible structural matching	Chan and Yeung [37]
Feature extraction and nearest neighbor classification	Chen and Yin [38], Fukuda *et al.* [39], Smithies *et al.* [13]
ART-based neural architecture and elastic matching	Dimitriadis and Coronado [40]
Hidden Markov model	Winkler *et al.* [41, 42, 43, 44], Sakamoto *et al.* [45]
Three-layered back propagation network	Marzinkewitsch [46]
Traditional template matching	Nakayama [47]

6.3 Stage 1: Data Collection and Normalization

This early processing stage is typically for the collection and normalization of handwriting samples. By performing simple noise reductions and correcting other irregularities in input such as skews or heavy data concentrations, it is possible to provide faster algorithms for character recognition. Another form of normalization that was found to significantly impact the amount of processing was slope conforming. By combing line segments whose difference in slopes was less than a pre-defined constant, Tablet PC and desktop stroke sizes were often reduced by half with little visual differences apparent.

Looking beyond handwriting recognition, Yu and Cai [48] use this stage of recognition to normalize line segments into approximate objects. In their attempt to create a *domain independent* recognizer, they utilize simple, low level geometric features including: lines, arcs, circles, ellipses and helixes. Blending the lines between stage 1 (data collection and normalization) and stage 2 (symbol recognition), Yu and Cai derived the following hypothesis on recognizing geometric features:

> Simpler is better which favors a smaller number of constituent primitive shapes, and more specific is better which prefers circles, ellipses and helixes than lines and arcs.

In blending the first and second stage processes, Yu and Cai are able to recognize increasingly complex objects, as seen in Figure 6-15.

With PDIA's primary purpose being a responsibility to provide a unified and complete inking experience, this stage provides two areas in which PDIA will be able to assist in math recognition: 1) Data Collection and 2) Ink Normalization.

Often prototyped recognition engines are responsible for data collection, ink manipulation and text recognition. Recognition engines built on top of PDIA will be able to take advantage of the existing ink functionalities, accepting as input only normalized ink in standardized containers. Developers of a recognition engine may safely assume ink will be *given* to them for recognition; they do not have to collect it from applications.

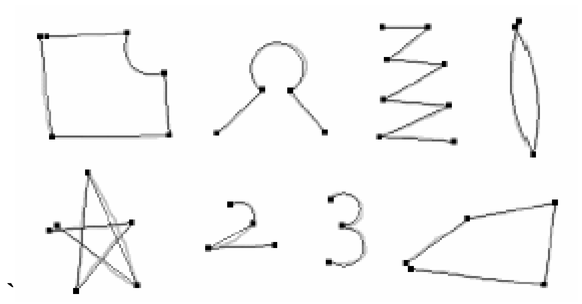


Figure 6-15 Examples of hybrid shapes with recognition results inYu and Cai's domain independent sketch recognition application.

6.4 Symbol Recognition

During the second stage of recognition, symbol or character recognition is responsible for detecting individual characters. Table 6-6 provides an introduction to various types of recognition methods and papers in which samples are provided.

If the segmentation stage has not already done so, this stage is responsible for piecing together or grouping related line segments, creating a Strokes object. For a more complete overview of the class hierarchy used in this thesis refer to section 4.3. A *set* of *Stroke* class objects, *Strokes* are created when individual line segments are thought to be related and grouped. While the first release of PDIA will not support auto grouping of line segments, it is possible that future researchers will improve on this functionality.

Simplistic yet effective methods of grouping line segments into *Strokes* include bounding box analysis, timing information, spatial proximity or a combination of these methods. In comparing bounding boxes, two *Strokes* whose bounding box overlap by a predefined percentage are assumed to be related. Using timing information, strokes that are drawn in quick succession are thought to be related, with a predefined limit of 1 – 3 seconds often representing a break in *Strokes*. Using spatial information, strokes that are nearby or within a predefined radius are thought to be related. Most frequently a combination of timing and spatial relationships is used when creating groups of line segments.

It will be necessary at this stage for most implementations of a math recognizer to reference an independent database for symbol, feature or other forms of comparisons. These databases typically include either graphical samples for Optical Character Recognition (OCR) or in the case of online recognition, ink data representing each character that is recognizable. For additional information on the database created by ORCCA of which I was a part, please refer to Section 8.7.

6.5 Structural Analysis

After recognizing individual characters, the next step of handwriting recognition is the construction of meaningful equations or expressions that represent the original intent. By identifying spatial relationships between symbols, it is possible to use parse or relationship trees to construct expressions that are meaningful in memory. If ambiguous equation is encountered, it will be necessary to acquire additional input from the user to resolve such uncertainty. We can then identify logical relationships between symbols and operators, further refining the output to an expression that follows predefined rules of mathematics. Many prototype recognition engines ignore this phase, Blostein and Grbavec [32] note that many researchers assume perfect symbol recognition, concentrating primarily on symbol arrangement analysis as seen in [49, 50, 51, 52].

After recognizing the spatial and logical relationships of symbols, it is possible to further analyze the overall structure of an expression often by examining the relative positions of symbols. By understanding the typographical identification of symbols, it is often possible to compare for other relative spatial relationships such as “in-lining, subscript, or superscript … we may further decide its corresponding association, namely, implicit multiplication, subscripting or exponentiation, respectively” [33]. Figure 6-16 illustrates expected typographical patterns of ascending, regular and descending symbols.

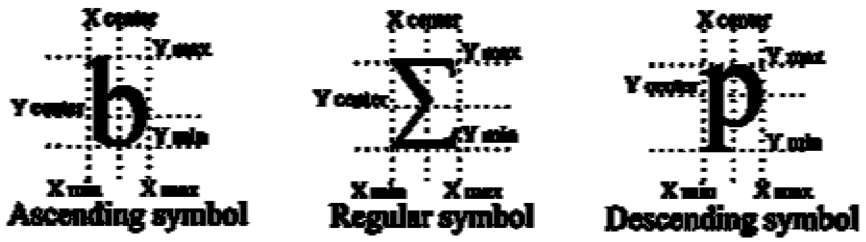


Figure 6-16 Typographical centers for different types of symbols

Structural analysis is the most independent stage of the mathematical recognition process. Because of its independent nature, it will likely exist as a modular plug-in that interact with PDIA only to acquire information related to ink strokes. If additional functionalities are added to PDIA beyond the work of this thesis, it is conceivable that simple API calls will assist in aiding contextual information construction. Examples of future API calls may include determining relative positions of the X & Y minimum, center and maximum properties of *strokes* objects. If such properties were added, it would likely remain the task of the recognizer to determine relative positions of symbols in contrast to others, while forming an overall picture of the input. PDIA will be able to help in ink manipulation, if further processing of ink is needed.

6.6 Context Analysis

A dictionary and grammar lookup is a common post-recognition step in handwriting recognition, significantly improving translation results. Microsoft has invested substantial effort to allow customizable dictionaries for the recognition of legal, medical and other specialized terminologies to allow high accuracy. For example, Figure 6-17 with symbol recognition returns only “nello vorld”. After running a post-recognition spell check, the intended results “hello world” is returned.

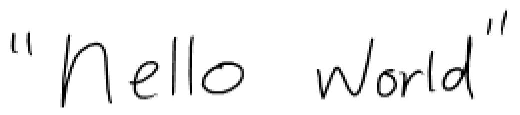


Figure 6-17 Ink collected by the Tablet PC. Once processed by a spell checker, it will be clear that the text should say “hello world”

The ability to perform a similar process on mathematical expressions is envisioned by ORCCA as a necessity for achieving highly accurate results. While the intrinsic details that make the creation of a "mathematical dictionary" are complex enough to become a thesis by itself, the concept is relatively simple. The creation of such a database has been already initiated, and members of the ORCCA lab have downloaded over four years, approximately 20,000 articles, of mathematical archives from the Los Alamos National Library[1] for dictionary prototyping. The next steps include creating or adapting an architecture that allows for the interior storage, archiving, indexing and traversal of mathematical expressions.

Upon completion, this dictionary will represent yet another application that will support PDIA, adding to the value of a mathematical framework and improving the success of eventual mathematical handwriting recognition engines.

6.7 Conclusions

In the creation of any online mathematical recognition engine, it is clear that significant dependencies exist on the underlying ink architecture. By creating a math API coupled with PDIA, substantial amounts of development not directly related to recognition is made modular. For instance, all ink manipulation such as normalization methods or property retrieval will become a part of the PDIA and math APIs. This allows a separation of inking and recognition APIs and a more modular framework. This organization will improve both the developer and end user experience, ultimately improving the recognition experience and contributing to the success of a recognition engine.

Lastly, we presented a commonly accepted three stage process of mathematical recognition. Data collection and normalization, Symbol recognition and Structural analysis categorize this process, along with ORCCA's vision of a fourth process: Context analysis, which is a post recognition process commonly used in recognizing string

[1] Downloaded from http://xxx.lanl.gov/archive/math, ORCCA lab members have downloaded and are processing all mathematical content available from January 2000 – July 2004

languages. It is our belief that the addition of context analysis will significantly improve our results in the same ways handwriting recognition has benefited from the additions of dictionary and grammar checks. Topped with an effective user interface that provides users with visual clues and immediate feedback, this fourth stage will hopefully provide visual language recognition results comparable to string languages.

Chapter 7 User Interface Requirements

7.1 Overview

As the purpose of this thesis is to identify the effects of a mathematical framework on the handwriting recognition of mathematics, it is appropriate to discuss additional means in which the math recognition process can be improved.

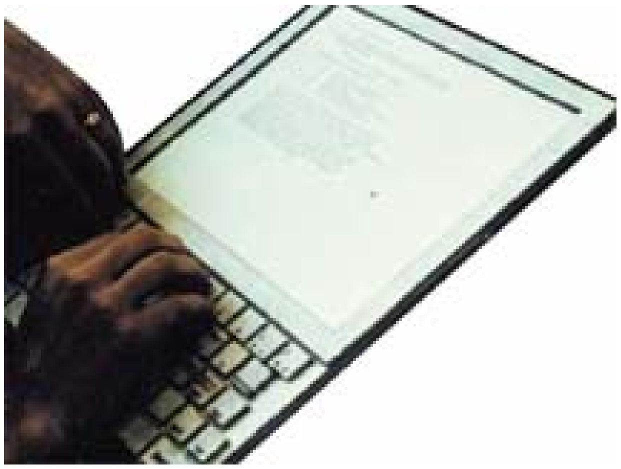


Figure 7-18 Dynabook Mockup provided by Larry Press [54]. While Kay's goal was a machine less than 2.5cm thick, the first version of the Dynabook, the "Interim" Dynabook was built using a desk-sized workstation.

This chapter outlines our expectation of the requirements necessary of a *User Interface* to be used in conjunction with our mathematical framework and a math recognition engine. Unlike Chapter 4, which discussed the PDIA architecture in great depth, this chapter is meant to provide an overview of how pen computing interfaces have evolved and what to expect of future interfaces. Through a review of past hardware and software interfaces and our own experiments with inking, we identify both requirements and suggestions in forthcoming applications that make use of our math framework.

Pen computing is not new: In 1945 Vannevar Bush [53] described a theoretical machine that would recall information of interest while recording thoughts, at the same time sharing information with others – all while using a pen like object for input. In 1975 Kay [54] envisioned the Dynabook as seen in Figure 7-18. The Dynabook was designed to use voice, pen and possibly a camera for input and output. Further emphasizing the importance of the pen as a method of input, Yourdon [55] prophesized about the future of computing in 1991 in saying: “I have seen the future and it is spelled P-E-N”.

What is new in recent innovations in pen computing is that the accompanying hardware over the past decade has made real time and online processing of high resolution pen computing possible. Continual improvements in hardware have provided commercial products to the public, at affordable prices with acceptable levels of input resolution and the CPU power necessary to process these high resolution input devices.

7.2 Text Based Mathematics Today

As stated in previous sections, the input of mathematics to a computer today is a cumbersome procedure. Users have a choice of two models: the graphical input / template method or through the use of any of several methods of text input. With substantial levels of differences in GUI based applications, and even variations in the text required for displaying versus solving mathematics, it is clear that a math recognition engine coupled with a proper user interface and stylus supported input as is needed.

Illustrating the choices available for entering text via mathematics, we present in Table 7-7, Table 7-8 and Table 7-9 a total of eight different methods of displaying to the screen the formula $\int_{\pi}^{2\pi} \sin(x)$. Table 7-7 illustrates three popular, full scale mathematical applications that are capable of solving complex equations. Instead of using a single industry standard for all three applications, each company has chosen to use proprietary notation that it believes is superior to other formats.

Table 7-8 illustrates the popular Microsoft Equation Editor, a component of the Microsoft Office suite of software. Useful only for displaying mathematics, users are forced to use a template system. While Latin characters are typically entered by the keyboard, all other symbols are entered with the mouse.

This technique is sufficient for many novice users; however more advanced users may find menus restrictive, navigation by mouse slow, and the ability to use third party plug-ins non-existent. If users require a more complete program, they are encouraged by Microsoft to purchase Design Science's MathType application where they must learn a new system of menu navigation. A more comprehensive solution, MathType offers advanced users the ability to export or input MathML or $L^{A}TEX$ [56], as well as the ability to choose between a user friendly menu driven mode or an expert based keyboard entry mode.

In addition to these four methods of input, if users are interested only in displaying mathematics on screen for print or manuscript creation, there are numerous options, four of which are displayed in Table 7-9. $L^{A}TEX$[1] is a popular choice for manuscripts, while Presentation MathML is often used in web browsers.

Table 7-7 Notational differences between three popular, high level, technical math solution engines for the formula $\int_{\pi}^{2\pi} \sin(x)$

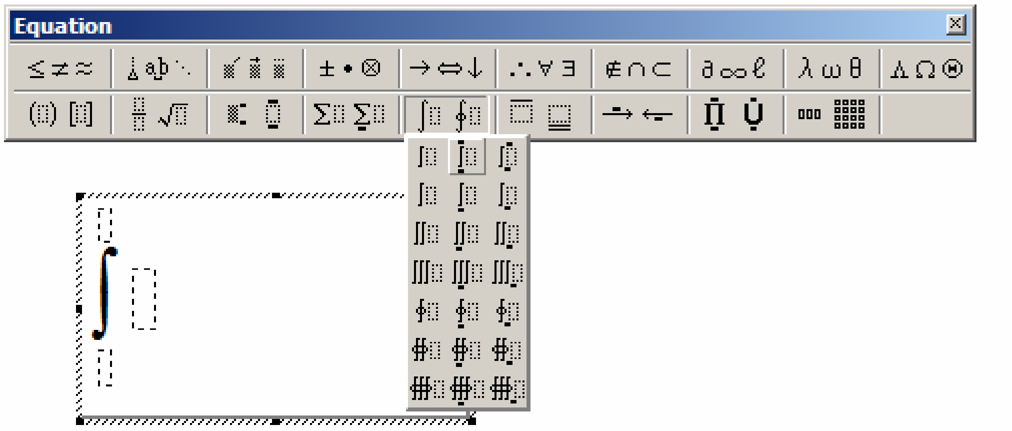
Maple	Mathematica	Matlab[2]
int (sin(x), x = 0 .. 2 *Pi)	Integrate [Sin[x], {x,0, 2* Pi}]	int (sin(x), x, 0, 2 * pi)

[1] $L^{A}TEX$ is a popular variation on TeX. Other variations include AMS-TeX and AMS-LaTeX

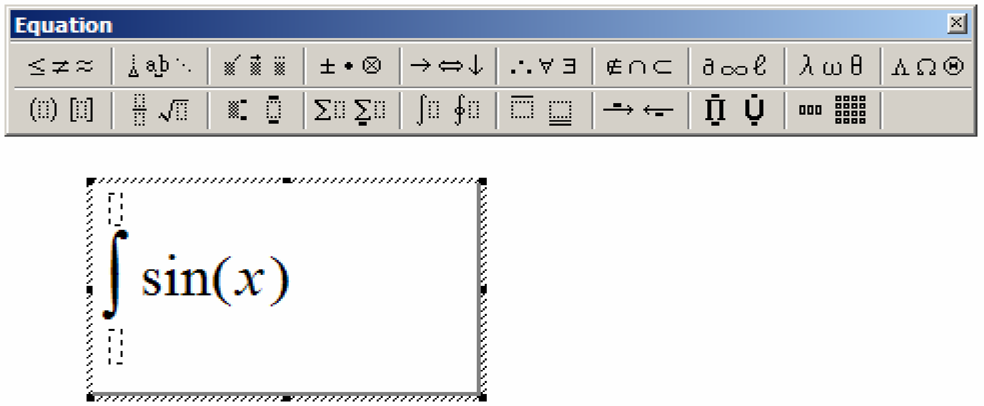
[2] There are several syntax options for Matlab, because it does not integrate, rather it applies a numerical method for estimating the integral. Therefore there are several commands, each one named after the method it implements.

Table 7-8 Screenshots and Instructions on how to use Microsoft Equation Editor to enter a common formula: $\int_{\pi}^{2\pi} \sin(x)$. Equation Editor is an add-on for Microsoft Office.

Step 1: User must find Integral template with correct upper and lower bounds



Step 2: User may enter Latin characters via keyboard



Step 3: Entry of Greek characters (π) must occur individually through menus

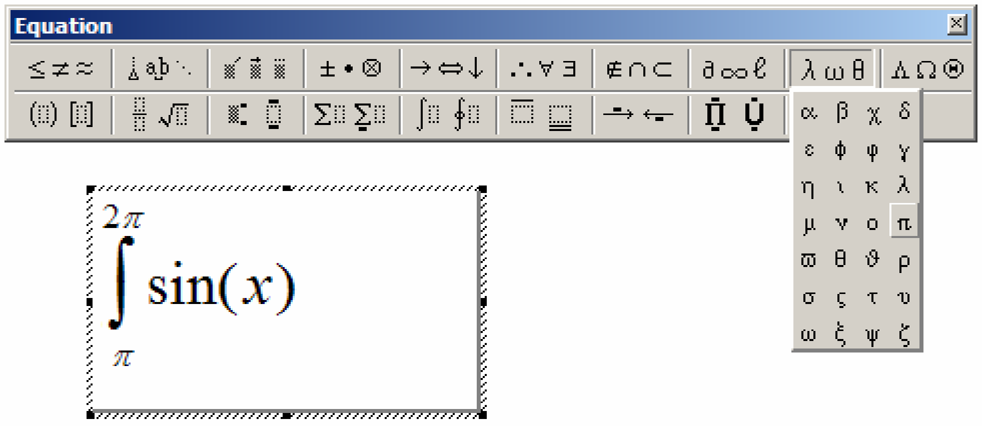


Table 7-9 Notational differences between four conventions used to display or otherwise present the formula $\int_{\pi}^{2\pi} \sin(x)$

$L^A T_E X$	Content MathML	Presentation MathML	Open Math

`\int_`
`{0}^{2 \pi} \sin(x)`
`{dx}`
	`<math xmlns=`
	`"http://www.w3.org/1998/Math/MathML">`
	`<apply>`
	`<int/>`
	`<bvar>`
	`<ci> x </ci>`
	`</bvar>`
	`<lowlimit>`
	`<cn> 0 </cn>`
	`</lowlimit>`
	`<uplimit>`
	`<cn> 2 </cn> < pi/>`
	`</uplimit>`
	`<apply>`
	`<sin/>  <ci> x </ci>`
	`</apply>`
	`</apply>`
	`</math>`
		`<math xmlns=`
		`"http://www.w3.org/1998/Math/MathML">`
		`<mrow>`
		`<munderover>`
		`<mo>&Integral;</mo>`
		`<mn>0</mn>`
		`<mn>2 </mn>`
		`<mo>&InvisibleTimes;</mo>`
		`<mn> &pi;</mn>`
		`</munderover>`
		`<mrow>`
		`<mi>sin</mi>`
		`<mo>&ApplyFunction;</mo>`
		`<mfenced>`
		`<mi>x</mi>`
		`</mfenced>`
		`</mrow>`
		`<mo>&InvisibleTimes;</mo>`
		`<mrow>`
		`<mo>&DifferentialD;</mo>`
		`<mi>x</mi>`
		`</mrow>`
		`</mrow>`
		`</math>`
			`<OMOBJ xmlns=`
			`"http://www.openmath.org/OpenMath">`
			`<OMA>`
			`<OMS cd="calculus1"`
			`name="defint"/>`
			`<OMA>`
			`<OMS cd="interval1"`
			`name="interval"/>`
			`<OMI> 0 </OMI>`
			`<OMS cd="nums1" name="pi"/>`
			`</OMA>`
			`<OMBIND>`
			`<OMBVAR>`
			`<OMV name="x"/>`
			`</OMBVAR>`
			`<OMA>`
			`<OMS cd="transc1"`
			`name="sin"/>`
			`<OMV name="x"/>`
			`</OMA>`
			`</OMBIND>`
			`</OMA>`
			`</OMOBJ>`

7.3 Early Requirements for Stylus Input Applications

One of the earliest recognized, yet still relevant requirements in pen computing is the need for visual feedback, ideally in real or near real time [57, 58]. This feedback could be obvious – the appearance of an ink stroke as the user moves the pen - or inconspicuous, such as smoothing of ink strokes as normalization occurs on user input. The Tablet PC is an example of both obvious and inconspicuous visual feedback; it provides users with a real time ink strokes and then performs normalization on the input, displaying a smoothed version of the input.

Hayashi *et al* [57] of Harvard developed a system for graphically inputting nonstandard (i.e., non-ASCII) or Chinese characters and recognized the following four advantages of a pen solution, they stated:

1) Nonstandard characters can be used without necessitating manual encoding arbitrarily assignment numerical or alphabetical codes
2) [Pen solutions] are flexible in that [they] can deal with any orthography or combinations thereof, allowing users to add custom symbols to the system's repertoire as encountered
3) Text may be proofread and edited as it is encoded and displayed on the screen
4) The text inputting component can be accomplished as quickly as the user can find [existing] characters or as quickly as characters can be inputted by hand.

Identified in 1968, these four benefits relate to mathematical input in the same way that they coincide with Chinese characters; entry was difficult and often requires arbitrary numeric codes, visual feedback was not always immediately available and the addition of new symbols was difficult. Today's mathematical software applications have the potential to significantly improve their user interfaces, should the stylus become an acceptable manner of input.

7.4 Stylus Supported User Interface Requirements

Another early solution, developed by Allen *et al* [58] of Yale University in 1981 was PEN[1] document editing system. PEN's purpose was "directed towards the preparation of manuscripts containing significant mathematical notation". At the time, it was recognized that document compilers like Scribe and TEX offered substantial customization in displaying mathematics, but lacked real-time capabilities. PEN provided real-time visual feedback by displaying the formatted document as it was typed, i.e. with specified fonts, visualizations of equations, or if a reference was made, by showing an icon of the reference instead of a number.

Recognizing that mathematics places unique requirements on software solutions, Allen *et al* provide us with a partial list of "desiderata for [the PEN] system", found in Table 7-10. Some of these criteria may be considered outdated (i.e.: support for a variety of alphabets and fonts) or keyboard specific (i.e.: it should not penalize too severely the entry of mathematical text). On re-examination of these criteria, it is clear how closely these requirements of math inputting via keyboard relate to a desirable math-pen computing experience as described in this thesis.

7.5 User Interface Requirements

Continuing the work by Allen *et al*, we use the seven requirements presented in Table 7-10 as key guidelines in creating our stylus based mathematical user interface. In this section, we reexamine these requirements and provide prototypes, mockups or examples

[1] Somewhat deceiving, the system entitled 'PEN: A Hierarchical Document Editor', actually uses a keyboard for input. Looking beyond this, the paper still identifies many of the problems related to displaying and editing mathematics onscreen.

of how these requirements will influence user interactions with a math handwriting recognition engine.

Table 7-10 Contrast of original PEN requirements and a theoretical application with similar goals but which uses a stylus for input.

Desiderata for PEN Document Editor, provided by Allen *et al* [58]	**Reexamination of Desiderata for Stylus Based Mathematical Input**
It should be interactive	Displays should be interactive with a stylus as primary input
It should not be unduly restricted by previous dependencies on paper as the display medium	Workspaces should not be restricted by previous dependencies on paper; display mediums may come in a variety of sizes and resolutions and user interfaces must take this into account
It should not penalize too severely the entry of mathematical text	Entry of mathematics should be mostly through free hand entry, and should not be too penalizing depending on system modalities
A character string representation of the text should be available for archiving and network transmission	Text representations of ink should be available for archiving and network transmissions
It should support programming and computation	Support programming and computation. For example, graph equations should generate (upon request) graphs by definition
It should be capable of adapting text to a reasonable output representation on paper	Capability should exist to adapt ink to displays with different output capabilities.
It should support a variety of alphabets and fonts.	Support for a reasonable variety of alphabets, symbols and writing styles.

7.5.1 Interactivity

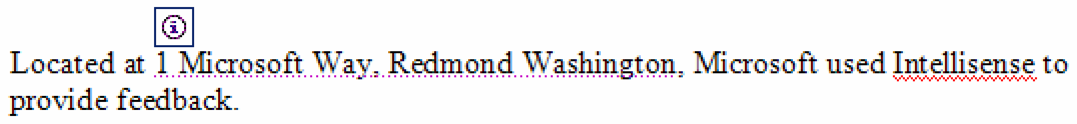
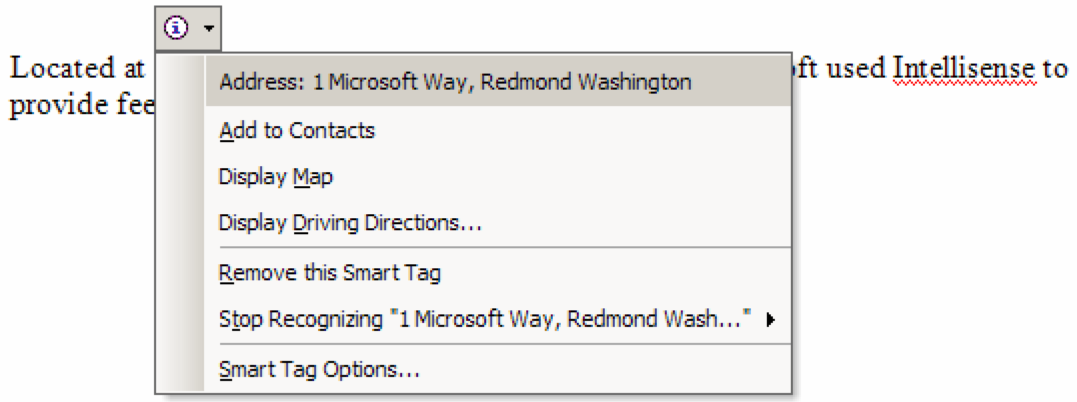
Introduced in section 2.6, the need for real time visual feedback was recognized as being important by several authors [20, 54, 55, 57, 58, 59]. Devices that are to be targeted by a math framework need to be capable of providing immediate visual prompts and cues to users, aiding in highly accurate recognition and creating a natural human computer interaction environment.

One form of immediate feedback used by Kurtenbach *et al* [60], and later marketed as one of the biggest improvements to Microsoft's Office Suite, was the concept of "SmartTags" or "Intellisense". These features were designed as an aftermarket plug-in for Office and their function is described below.

To illustrate, one could type an address in Microsoft Word, which recognizes a "place" event occurred. Table 7-11 illustrates just this occurring; an address is entered into a Word document, which then immediately provides feedback on the address. Other SmartTags provided by Microsoft include: Date, Financial, Names, Places, Telephone Numbers, and Time. Another form of Intellisense is also illustrated in Table 7-11; the word "Intellisense" is not found within the default dictionary and is underlined in red. This provides immediate feedback to the user alerting them to possible spelling mistakes.

It is foreseeable that a math based application would provide a similar experience for allowing users to confirm recognition results. A SmartTag like function would allow users to choose the correct recognition output, resulting in the meaning of such ink translations to become static or fixed, based on user input.

Table 7-11 Illustration of the Microsoft Address Smart Tag and the options presented to the user

Event 1: User enters an Address, Place, Landmark, etc.  Located at 1 Microsoft Way, Redmond Washington. Microsoft used Intellisense to provide feedback.
Event 2: Application provides user with a list of actions related to the address.  Located at … ft used Intellisense to provide fee… Address: 1 Microsoft Way, Redmond Washington Add to Contacts Display Map Display Driving Directions... Remove this Smart Tag Stop Recognizing "1 Microsoft Way, Redmond Wash..." Smart Tag Options...

Another form of visual feedback is the use of subtle clues to guide users. Figure 7-19 illustrates an example of a recognized integral symbol. The user is then presented with clues of where to enter the parameters. This visual aid is one means of limiting the use of soft conventions, which were identified asproblematic in section 5.1. By providing users with a restrictive environment, the use of visual clues could greatly improve accuracy with little inconvenience to users. Another visual clue used in Figure 7-18 is the use of blue horizontal lines. These lines are reminiscent of those found in elementary school work books, and are common place in natural language recognizers. Although subtle, they guide users and allow them to write in more level lines, although with mixed success. This allows for increased accuracy for recognizers, by providing an approximate baseline that is more easily identified, as shown in Figure 7-19.

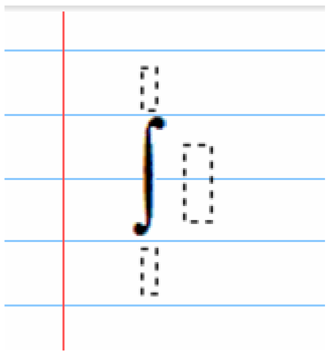


Figure 7-19 Recognized Integral with visual clues of where to enter parameters

7.5.2 Display Medium Restrictions

Moving away from the 8.5” x 11” sheet of paper, a user interface should not penalize the user for not having a display of sufficient size. For instance, when deployed to devices with smaller screens such as a Pocket PC, the interface may resemble mini whiteboards or blank slates, with few options or visual clues as screen real-estate becomes more precious. On devices with larger screens such as a Tablet PC, a complex variety of visual clues may be presented to the user. These may include options to set mathematical domain information or an area that is dedicated to displaying optional recognition results, allowing the user to easily choose the correct result.

7.5.3 Entry of Mathematics

Entry should be provided either through a freehand, template free whiteboard or via a template-based solution which acknowledges the use of domain information as provided by the user.

Used by Lank *et al.* [61] in the creation of their domain specific UML recognizer, it is understood that a "domain specific refinement [enables one] to achieve highly accurate recognition results" [61]. While it should not be required, it would be useful if the user provided the UI with domain information based on the type of mathematics entered. For instance, domains will skew post recognition results by comparing input against commonly used formulae of the respective domain, as in the ongoing work by So, Watt *et al*.

7.5.4 Persistent Storage and Transmission of Data

Any application that makes use of PDIA will have a means of saving ink information. For details on PDIAs persistent storage mechanisms, please refer to section 4.4. Once saved to disk, an application could use any network protocol to transfer information over a network. Furthermore, because of the use of InkXML, other applications will be able to easily interoperate with ink data produced by PDIA.

7.5.5 Support for Programming and Computation

To be truly interactive, a handwriting user interface for math could allow users to solve mathematics as it is recognized. For example, Figure 7-20 illustrates the input of an equation (above) with the solution provided (below).

These interactive capabilities will involve creating or partnering with one of the many mathematical engines such as Maple, Mathematica or Matlab. Historically, ORCCA's affiliation with MapleSoft suggests future products will use their software to drive any computation of mathematics involved.

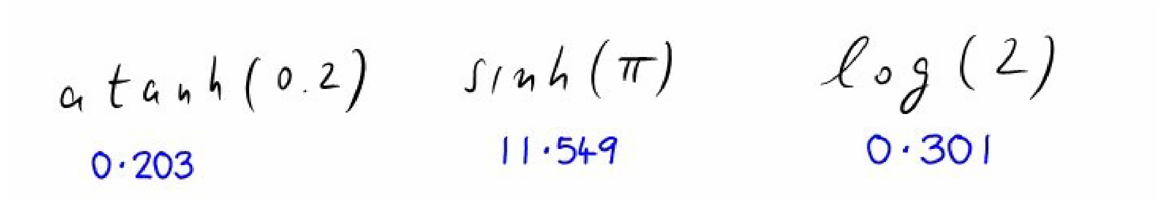


Figure 7-20 After entering a function, users will expect a feature rich application to provide them with the computational result to the provided equation

7.5.6 Output of Mathematics

In addition to being capable of transmitting data, a UI should provide a means of adapting and optimizing ink and accommodate the limitations of various input devices. For instance, when moving ink from a Pocket PC to a Tablet PC, several screens worth of data may be capable of being displayed at once. A UI should take advantage of this, displaying as much ink as possible without scaling or otherwise modifying the original ink beyond pre-defined criteria.

7.5.7 Support for Symbol and Handwriting Variations

Any mathematical handwriting recognition application that is fully functional will have to address extremely large symbol sets, as well as variations in each symbol. It will be the responsibility of the recognition engine to accommodate each domain and its respective symbol set, although the UI may request domain information from the user.

7.6 Conclusions

The seven requirements presented in sections 7.5.1 to 7.5.7, as well other suggestions provided in this chapter, may help ensure users will have a complete end to end solution for handwriting mathematics. Users could take notes on a Pocket PC, transmit over a network to a Tablet PC or desktop, further refine and ultimately print finished works on paper or to a manuscript for electronic distribution.

It is the ability to provide this complete solution that will ensure computer-based mathematical handwriting recognition gains acceptance. This acceptance will further drive the need for a comprehensive math framework, as outlined in this thesis.

Chapter 8 Implementation and Experiments

8.1 Overview

This chapter presents challenges in implementing the inking architecture and introduces our experiments and tests. These include creating a survey to collect handwriting recognition and implementing a Bentley-Ottmann line sweep algorithm

This chapter also presents other notable conclusions from our smaller experiments and prototypes that were done during the main study. These include implementation decisions, extending to the choice of interpreted language such as Java and C#, user interface and survey questionnaires as well as questions on the importance of efficiency.

8.2 Priorities of Properties within a Math Framework

As in any project with identified properties, it was important to assign priorities to the individual requirements of the math framework discussed in this thesis. Each requirement presented offered a potential advantage for the framework.

While it was difficult to assess the priority of each requirement, it was clear that before a recognizer could be implemented, a working ink environment must exist. While prototypes for inking have been created twice before at ORCCA, we felt it was necessary to begin work on a production quality API that would allow the progression of research in other areas. The first of these prototypes was a proof of concept based on an older model IPAQ Pocket PC. The second was a proof of concept that illustrated the ability to generalize ink data collected either from the Tablet PC or an IBM CrossPad into a common format. Results from both prototypes helped in the implementation of this thesis.

The creation of a functional inking experience has been the primary focus during testing and implementation of this thesis. Using the requirements identified in Chapter 3 and the architecture presented in Chapter 4, we have implemented the necessary class objects: *Point, Line, Rectangle Stroke and Strokes.* Functionally, this has allowed us the opportunity to permit a nearly complete inking interface on the desktop, Tablet PC and Pocket PC, and to study how they can be related.

8.3 Choosing a Language

In order to create a platform independent of processor or operating system, C++ was the language decided upon for low-level code. If limited to the standard template libraries and ANSI / ISO specifications, C++ proves to be flexible as well as portable, with compilers available for most hardware platforms of interest. C++ is also capable of being extended by both Java and C#, allowing future development to take advantage of these environments. By strictly adhering to standard C++ specifications, often all that was required to port code to another platform was to recompile the code; this was true for Windows XP to Linux as well as Windows XP to Windows Mobile (Windows CE). The standard template libraries are the only dependencies on external libraries in PDIA.

Given that our application is by design CPU intensive, it was important that as little overhead as possible be added to our software. At present, C++ requires less processing power than an interpreted language such as Java or C#. When comparing C and C++, the object oriented nature of C++ proved to be better suited then using C alone.

8.4 Building for Individual Platforms

Given the hardware requirements identified in section 2.6 and the recommended software platforms described in section 2.7, it was decided that we would implement PDIA on three major platforms: The Windows XP Tablet (Tablet PC), Widows XP (desktop) and Windows Mobile (Pocket PC). These three platforms were chosen for their representative nature, ease of deployment and broad user base. The Palm, Apple and Linux platforms could be address at a later date.

These platforms are related to each other, in regard to PDIA implementation. The Tablet PC runs a superset of Windows XP called Windows XP Tablet edition. It is a complete version of the Windows XP operating system with additional support for inking. Any application compiled on a desktop will also run on a Tablet PC. Surprisingly, the desktop and Pocket PC also have one common property: neither platform has any native inking functionality that is available for developers to use. Therefore an application built for the Pocket PC will require as much custom implementation as a desktop version.

8.4.1 Build Environment

Wanting to produce a maintainable project, the logical layout of each module on disk is illustrated in Figure 8-21. This screen shot (from within Visual Studio.Net 2003) illustrates the directory layout used to ensure each module of PDIA can be easily traced, ensuring maintenance and extension is as simplistic as possible.

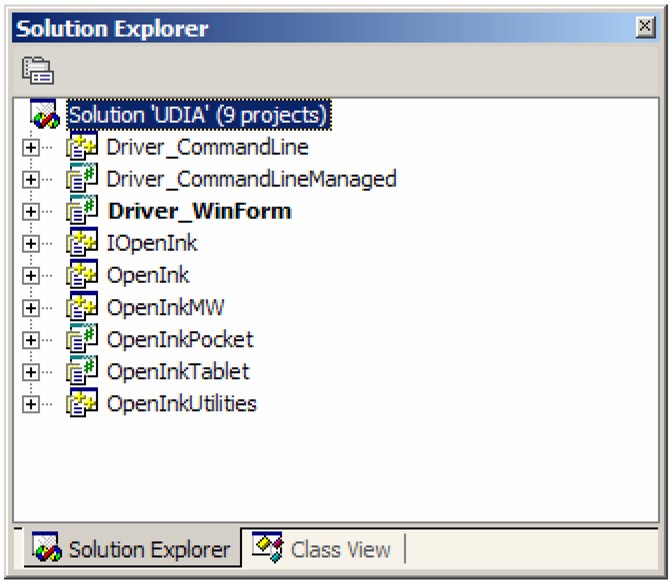


Figure 8-21 Implementation hierarchy from within Visual Studio.Net 2003

8.4.2 Tablet PC

For PDIA to operate on a Tablet PC, the only requirement was to use the well defined interfaces and create thin wrappers over existing ink functionalities. Table 8-12 illustrates an example of returning a bounding box of an *Ink* object from the Tablet PC

SDK. As seen in Table 8-13, our implementation of the *Ink* object will take advantage of the Tablet PC SDK, acquiring the bounding box using the native Tablet PC SDK.

Table 8-12 C# code that utilizes the Tablet PC SKD to acquire the bounding box of an *Ink* object

```
// Returns the bounding box of an Ink Object
System.Drawing.Rectangle BoundingBox(Microsoft.Ink.Ink inputInk)
{
   System.Drawing.Rectangle inkBB = inputInk.GetBoundingBox();
   return inkBB;
}
```

Table 8-13 PDIA Managed C++ code that determines the bounding box of an *Ink* Object

```
// Returns a rectangle representing the bounding box of the stroke
__property mRectangle * get_BoundingBox()
{
   // Note, mSession is a wrapper to a managed pointer to the Tablet
   PC's ink object.
   return new mRectangle( mSession->BoundingBox() );
}
```

In the previous example, it is seen that there is additional overhead to the PDIA implementation existing on top of the Tablet PC API. In more complex operations, this overhead is negligible. In section 8.10 the cost and difficulties in computing intersections are discussed, requiring a runtime of $O(n \log n + k)$, where n is the number of points in a stroke and k the number of intersections. This computational cut dominates any $O(n)$ cost associated with a wrapper layer.

8.4.3 Desktop PC

With no native ink support, creating an inking environment for the desktop required a complete, custom implementation of all class objects, described in Chapter 4.

Creating an ink object required capturing mouse down clicks, sequential x, y coordinates and storing this captured data within a custom implementation of a *Stroke* class. By building a complete implementation with no dependencies for the desktop environment, we ensure all other platforms can support PDIA, even if they simply use a copy of this platform's implementation. This implementation could serve as a starting point for any other platform.

8.4.4 Pocket PC

Similar to the desktop, Pocket PCs do not have much in the support of interactive inking API beyond support for creating *point* objects. As stated in section 8.4.3, much of the implementation for the Pocket PC is identical to the desktop. The only changes are in the default reserved sizes for underlying data structures – for instance a Pocket PC is expected to collect significantly less data per *Stroke* object than a desktop.

8.5 Extending to the Microsoft .Net Framework

To take advantage of the Microsoft .Net framework support for rapid prototyping of graphical user interfaces, PDIA was extended to support the Microsoft Common Language Runtime. This involved using Microsoft's "managed" extensions to C++ to create a dynamically linked library (dll) capable of interoperating with both native C++ code as well as code on the .Net framework. This is essentially an interface between the two programming paradigms.

Table 8-14 illustrates a complete yet simple[1] example of how one could use native C++ classes, extend them with Microsoft's managed extensions for C++ and ultimately use this code in a C# application. The example in Table 8-14 defines an interface for a *Point* class that is then implemented in native C++. Using managed extensions for C++, the native class is referenced through a pointer, using appropriate methods to expose its original functionalities. This managed class can be used in any language supported by the .Net framework.

[1] While Table 8-14 does include all code necessary to extend a C++ object unto the Microsoft .Net Framework, additional configuration and environmental settings have to be changed. Please refer to MSDN articles 814472, 148652 and "Converting Managed Extensions for C++ Projects from Pure Intermediate Language to Mixed Mode".

8.6 Extending to Java

Using either Sun Microsystem's *Java Native Interface* (JNI) or Microsoft's *Raw Native Interface* (RNI) it is possible to use native C++ code onto the Java runtime environment. The differences between JNI and RNI are found in the libraries provided by each to provide a *Native Interface* and in the ways each is implemented. The Java code will be similar in both cases, but the C++ code will be slightly different. Sun provides "jni.h" library and Microsoft the "native.h" library. Both, methods use similar concepts although the syntax is different. Furthermore, Microsoft's implementation does not hide implementation details whereas Sun's implementation uses abstractions to prevent exposing this information.

This thesis does not preclude extending PDIA to Java, providing hardware and software requirements are tailored to accommodate any additional needs the Java runtime may require. In implementing the concepts presented in this thesis we did not produce Java code that was satisfactory or useful, so we leave this work to future colleagues working on PDIA.

Table 8-14 Illustration of how to extend Native C++ code with Managed Extensions for C++. Once extended to the Microsoft .Net Frame, all supported languages such as C# or Visual Basic.Net can make use of the original c++ classes.

Native C++ Code: Interface & Implementation for a Point Class

```
class IPoint
{
public:
   virtual float X() const = 0 ;    // Returns X Coordinate
   virtual void X(float x) = 0 ;    // Sets X Coordinate
   virtual float Y() const = 0 ;    // Returns Y Coordinate
   virtual void Y(float y) = 0 ;    // Sets Y Coordinate
};

class Point: public IPoint
{
protected:
   float _X, _Y         // The values representing X & Y

public:
   // Creates an Point object with default coordinates (0,0)
   Point(void) :  _X(0), _Y(0)  {}

   // Creates an Point object with specified coordinates
   Point(float x, float y) :  _X( x ), _Y( y )  {}

   // Copy Constructor
   Point(const IPoint &pt) :  _X( pt.X() ),  _Y( pt.Y() )  {}

   float X() const      {  return _X;     }
   void X(float x)      {  _X = x;     }
   float Y() const      {  return _Y;     }
   void Y(float y)      {  _Y = y;     }
};
```

Managed Extensions for C++: Extending the Point Class for use on the Microsoft .Net Framework

```
// Allow use of managed objects from the .Net Framework
using namespace System;

public __gc class  mPoint :
// Necessary for Copy Constructors on .Net Framework
   public ICloneable {

private:
   Point __nogc * mSession;      // An unmanaged Point object

public:
   // Creates an mPoint object with default coordinates (0,0)
   mPoint(void) :  mSession( new Point() )  {}

   // Creates an mPoint object with specified coordinates
   mPoint(float x, float y) :  mSession( new Point(x, y) )  {}
   // Creates an mPoint object referencing an existing Point object
   mPoint(Point * pt) :  mSession( pt )  {}

   // Copy Constructor, as defined by the interface ICloneable
   virtual Object* Clone()
   {
      mPoint * managedPoint = new mPoint;
      *( managedPoint->mSession) = *mSession;
      return managedPoint;
   }

   // Destructor
   ~mPoint(void)  {  delete mSession;  }

   // Returns a non-Garbage collected Point object
   __property Point __nogc * get_PointPtr()  {  return mSession;  }

   // returns the value representing the X coordinate of the point
   __property float get_X()  {  return mSession->X();  }

   // Sets the value representing the X coordinate of the point
   __property void set_X( float value )  {  mSession->X(value);  }

   // returns the value representing the Y coordinate of the point
   __property float get_Y()  {  return mSession->Y();  }

   // Sets the value representing the Y coordinate of the point
   __property void set_Y( float value )  {  mSession->Y(value);  }

};
```

C#: Using Managed C++ Object in C#

Note that C# or any other language supported by the .Net framework can use the class above, mPoint. No special syntax or steps are required, beyond manually adding a reference to mPoint.

```
using System;

namespace CSharp_Execution_of_Managed_CPP
{
   class MainDriver
   {
      [STAThread]
      static void Main(string[] args)
      {
         mPoint p1 = new mPoint(9,9);
         mPoint p2 = new mPoint(100,100);
      }
   }
}
```

8.7 Data Collection Survey

Available in its entirety in Appendix A, the math survey was first created by ORCCA in 2002 by Dr. Stephen Watt and Xiaojie Wu as a means of surveying individuals to collect mathematical ink samples on the IBM CrossPad. It was revised during the timeframe of this thesis by myself to collect ink surveys through the Tablet PC.

The survey is broken down into seven sections, including samples of:

1) Alphanumeric
2) Latin
3) Greek
4) Script
5) Simple Formulae
6) Complex Formulae
7) Matrices.

This comprehensive survey takes approximately 20 minutes to complete. It collects 301 unique symbols as well as, 68 formulae and matrix samples. On the Tablet PC each survey generates approximately 1 MB of data. The types of data recorded by the questionnaire are explained in Table 8-15.

Currently, there have been two stages of ink collection surveys, the first by Ben Huang with the IBM Crosspad and the second by myself on the Tablet PC. A total of approximately 40 surveys were captured on each of the IBM Crosspad and an Acer Tablet PC. Because of the nature of the device, those collected on the Crosspad include significantly less data than the Tablet, and do not include pressure, off screen coordinates or detailed timing information. Having data from a “poor” device as well as a “rich” device is useful in designing a cross-architecture framework.

The Tablet PC surveys are responsible for creating 15,000 samples of ink, or almost 90 MB total in raw data. Using some of the recognition methods illustrated in Table 6-6, including elastic matching and feature matching, this database of ink collected is ORCCA's primary source of known ink samples for mathematics.

Table 8-15 Type of data collected by Tablet PC version of the ORCCA Ink Survey

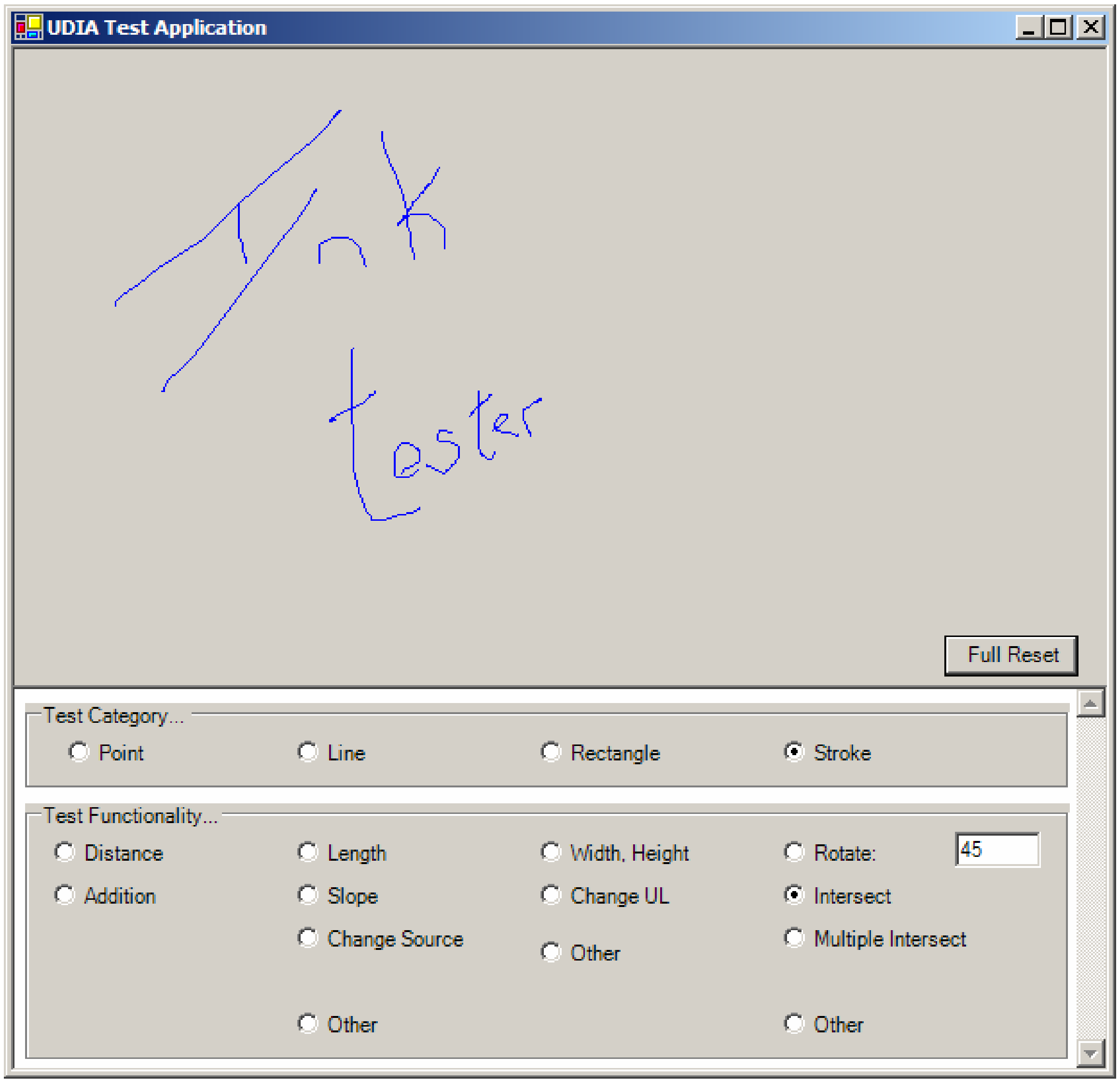
Data Type	Description

X, Y Coordinates	Every X, Y coordinate reported by the stylus was recorded during the survey, including those reported when the status was 0, or the pen was off the screen. The algorithm used to record coordinates was as follows:
	if (stylus touches screen) { then record coordinates }
	While (stylus is within ink-able area) {
	    record coordinates
	    if (stylus leaves ink-able area)
	        break & mark area as non-ink-able
	}

Stylus Status	1 or 0, toggled for the stylus being on or off of the screen. A 1 was recorded when the stylus was on the screen, 0 otherwise
Pressure	The Acer Tablet PC used during collection was capable of 255 levels of pressure. An integer representing the pressure was recorded with each X, Y coordinate. This information may be used in discovering outlying data points, such as the beginning or end of a stroke.
Timing	At the beginning and end of each stroke, a timestamp was made.

8.8 User Interface Experiments

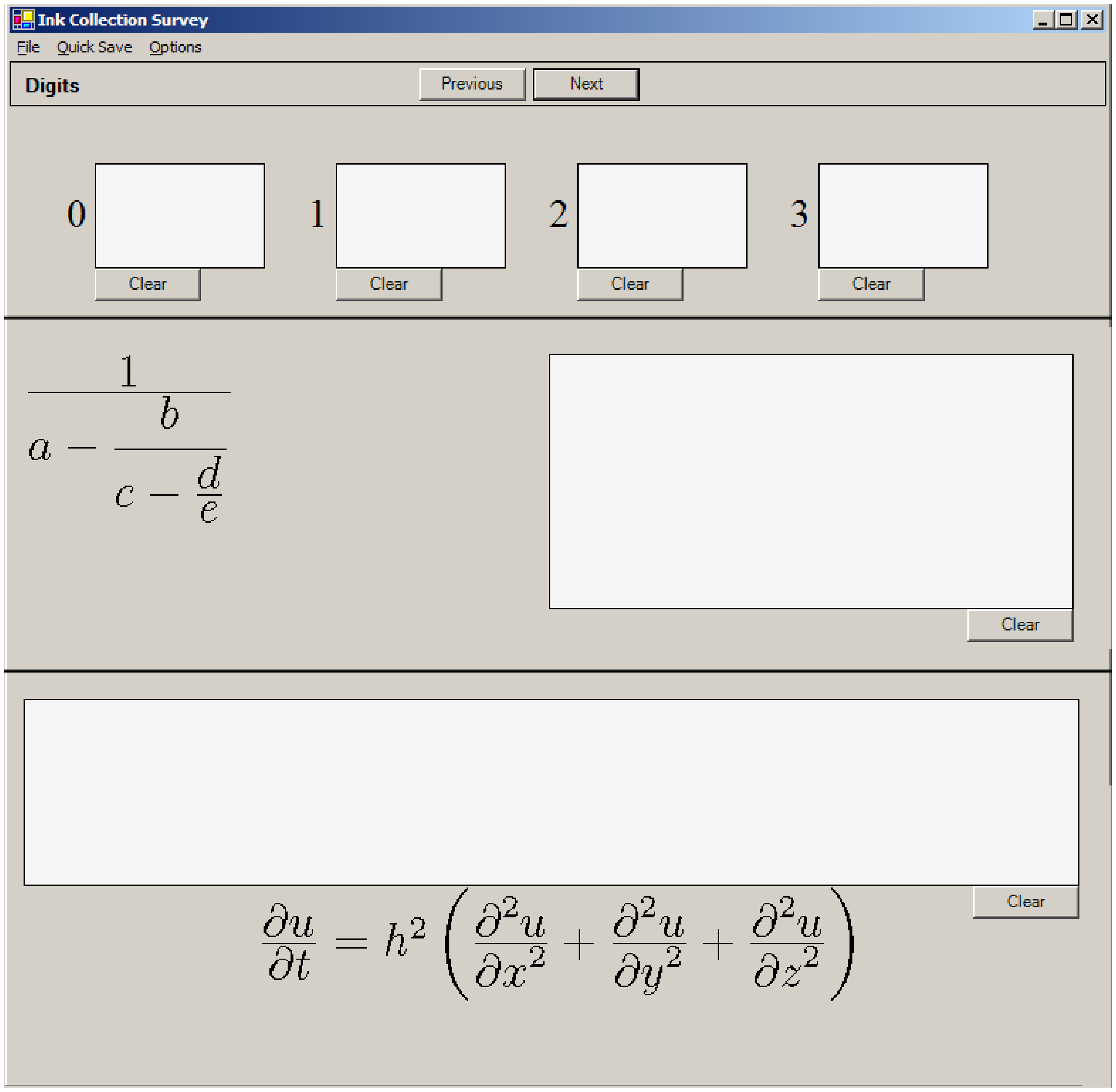
Focusing on creating a functional ink architecture, two limited purpose interfaces were created to assist the creation of our math framework. As seen in the screen shot of Figure 8-22, the first was for testing and debugging PDIA. The second interface, as seen digitally altered in Figure 8-23, was used to survey individuals for data collection purposes. Neither of these interfaces addressed specific requirements necessary for a math framework, although both interfaces provided results beyond initial purposes.



Test Category...
Point Line Rectangle [✓ Stroke]

Test Functionality...
Distance Length Width, Height Rotate: 45
Addition Slope Change UL [✓ Intersect]
Change Source Other Multiple Intersect
Other Other

Figure 8-22 Screen shot from the PDIA test application. Designed only as a means of testing inking functionality, it was noticed in this application that putting menu driven functionality below the inking experience was more convenient than above.



$$\frac{1}{a-\frac{b}{c-\frac{d}{e}}}$$

$$\frac{\partial u}{\partial t}=h^2\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right)$$

Figure 8-23 A representative, digitally composed image comprising of three sections from different pages of the ORCCA ink collection survey. Each section in this image is separated by a horizontal line, which was not a part of the survey application.

8.8.1 Ink Tester: User Interface Results

The screen shot provided in Figure 8-22 demonstrates the application that was used to test our PDIA implementation. Designed to provide a means of testing the functionality of each class object, it also yielded results that are beneficial to future user interfaces.

It was discovered with this user interface that important menu driven commands are more conveniently placed at the bottom of the screen. While placing menus at the bottom of the screen is convenient, it is also against commonly accepted practices in computing. It appears that in this setting ease of hand movement is more important than visual organization. The acceptable UI practice is that menu items at the bottom of the screen have the potential to create a disjointed or confusing user experience. Instead, it would be possible to use the bottom of the panel for other important tasks.

8.8.2 Questionnaire: User Interface Results

The screen shot in Figure 8-23 is a representative image, combining sections from three pages of the math collection survey into one image. In the image, each of the three regions is separated by a horizontal black bar. Notable about this survey is that while inking, people tend to require more space on a computer than on paper. Other conclusions include:

8) Individual characters were reproduced at a size comparable to a size 24 point font on a 14” display with 1024 x 768 resolution
9) Formulas required approximately 50% more space horizontally to reproduce, but only 25% more space vertically

When reproducing large formulas from a sample, having the sample formula below the inking field was easier for most right handed people. Left handed people found it more convenient to have the larger sample formulas on top of the inking panel.

8.9 Efficiency of PDIA Implementation

It was important during development of PDIA that efficiency be considered at all times. It would not be unusual to work with thousands of point segments at any given moment and require real-time response.

Together the *Point,Line* and *Rectangle* classes are clearly capable of executing all functionalities in constant time; the amount of data contained within each object is constant. It is also true that certain functions within these classes require more CPU cycles than others: returning the X coordinate of a *Point* versus rotating a *Rectangle* or finding the intersection point of two *Line* objects.

Unlike the foundational classes (*Point*, *Line* and *Rectangle*), the more complex *Stroke*, *Strokes* and *Ink* classes hold a variable amount of data. With *Ink* depending on the number of contained *Strokes,* and *Strokes* depending on the number of contained *stroke* objects, both of these objects' runtimes depend directly on the *Stroke* class. Because of the impact of the *Stroke* class on the efficiency on PDIA and the math framework overall, it is critical that all functionalities of the *Stroke* object are implemented efficiently. Table 2-2 illustrates the run-time complexities of every function within our implementation of the *Stroke* object for PDIA.

Table 8-16 Sample of public functions within the *Stroke* class along with respective worst case runtime

Function Prototype	Runtime (worst case)
// Constructors	O(1)
Stroke(void);	O(1)
Stroke(int initalSize);	O(1)
Stroke(const Stroke &copy);	O(1)
virtual ~Stroke(void);	
// Properties	O(1)
int ID() const;	O(1)
void ID(int newID);	O(1)
void addPoint(const IPoint & newPoint);	O(1)
void removePoint(int index);	O(n) or O(1)*
Rectangle * BoundingBox() const;	O(1)
int PointCount() const;	O(1)
vector<IPoint *> * Points() const;	O(1)
Point * ReturnPoint(int index) const;	O(1)
bool Deleted() const;	O(1)
void Deleted(bool newFlag);	O(1)
bool PenDown() const;	O(c)
void PenDown(bool newFlag);	
// Methods	O(n)
void clip (const IRectangle & newBoundingBox);	O(n)
void rotate(float angleDeg);	
void rotate(float angleDeg, const IPoint &	O(n)
centerPt);	O(n)
void scale(float scaleX, float scaleY);	O(n)
void scale(const IRectangle & rect);	O(n)
void smooth();	O(n)
void shear(float angleDeg);	O(n)
void deslant();	O(n)
void interpolate();	O(n)
void resampling();	
// Complex Methods	
void Intersection(vector<Point *> & iPts)	O(n log(n) + k)
const;	
void Intersection(vector<Point *> & iPts,	O(n log(n) + k)
const Stroke * s) const;	
* If the bounding box has already been found, runtime is constant. If the stroke has been modified in any way, runtime is O(n)	

8.10 *Finding Intersection Points*

Unlike the intersection between two line segments, which can be done in constant time, finding the intersection between two *Stroke* objects is the most CPU intensive algorithm we have implemented. Because *Stroke* objects are not a solid interconnected stream of data but rather a series of sample points, it is not sufficient to check for overlapping data points. Figure 8-24 illustrates the word “hello” by indicating data points with red dots. Although only a representative image, it is clear only checking overlapping data points is not a reliable means of finding intersections.

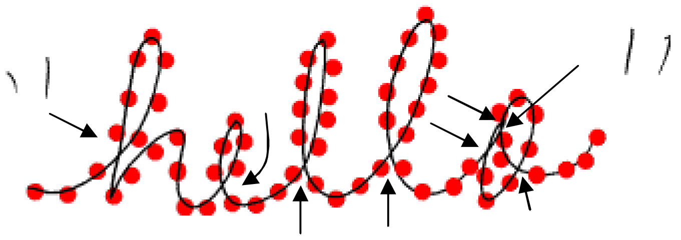


Figure 8-24 A representative image illustrating the sequences of data points that once connected by a solid line, constructs the word "hello", with arrows indication intersection points

8.10.1 Brute Force Intersection Algorithm

Clearly effective and simple to implement as shown in Table 8-18, a brute force method has a runtime of O(n^2). This is unacceptable for a critical function for use in handwriting recognition.

8.10.2 Bentley-Ottmann Line Sweep Intersection Algorithm

Accepted as optimal and proven in [62, 63], the Bentley-Ottmann line sweep algorithm uses contextual information to determine which line segments could possibly intersect at any given time, and only checks against these segments for intersection points. With appropriate data structures, it is possible to execute this algorithm in time O(n log(n) + k). This is illustrated in Table 8-17. For details in implementing this algorithm as well as a complete analysis of its complexity, see [63].

Table 8-17 Pseudo code and respective runtimes for using the Bentley-Ottmann line sweep algorithm to find self intersection points within a *Stroke* object

```
Pseudo code for Bentley-Ottmann Line Sweep Algorithm      Runtime

Bentley_Ottmann {
    Initialize event queue x = all segment endpoints;     O( n )
    Sort x by increasing x and y;                         O(n log n)
    Initialize sweep line SL to be empty;                 O( 1 )
    Initialize output intersection list L to be empty;    O( 1 )

    While (x is nonempty) {                               O( n )
        Let E = the next event from x;                    O( 1 )
        If (E is a left endpoint) {
            Let segE = E's segment;                       O( 1 )
            Add segE to SL;                               O( log n )
            Let segA = the segment above segE in SL;      O( 1 )
            Let segB = the segment below segE in SL;      O( 1 )
            If (I = Intersect( segE with segA) exists)    O( 1 )
                Insert I into x;                          O( log n )
            If (I = Intersect( segE with segB) exists)    O( 1 )
                Insert I into x;                          O( log n )
        }
        Else If (E is a right endpoint) {
            Let segE = E's segment;                       O( 1 )
            Let segA = the segment above segE in SL;      O( 1 )
            Let segB = the segment below segE in SL;      O( 1 )
            Remove segE from SL;                          O( log n )
            If (I = Intersect( segA with segB) exists)
                If (I is not in x already)
                    Insert I into x;                      O( log n )
        }
        Else {  // E is an intersection event
            Add E to the output list L;                   O( k )
            Let segE1 above segE2 be E's intersecting
                segments in SL;                           O( 1 )
            Swap their positions so that segE2 is now
                 above segE1;                             O( 1 )
            Let segA = the segment above segE2 in SL;     O( 1 )
            Let segB = the segment below segE1 in SL;     O( 1 )
            If (I = Intersect(segE2 with segA) exists)
                If (I is not in x already)
                    Insert I into x;                      O( log n )
            If (I = Intersect(segE1 with segB) exists)
                If (I is not in x already)
                    Insert I into x;                      O( log n )
        }
        remove E from x;
    }
    return L;
}
```

Table 8-18 Pseudo code and respective runtimes for using a brute force algorithm to find self intersection points within a *Stroke* object

Pseudo code for Brute Force Intersection Discovery	Runtime
`Brute_Force {`	
`    For Each (seg1, line segment in Stroke ) {`	`O( n )`
`        For Each (seg2, line segment in Stroke ) {`	`O( n )`
`            If (I = Intersect(segE2 with segA) exists)`	`O( c )`
`                Add I to the output list L;`	`O( c )`
`    }    }    }`	
`    return L;`	
`}`	

8.10.3 Conclusions in Intersection Point Algorithms

The graphs in Figure 8-25 and Figure 8-26 illustrate the measured time to compute intersections against the number of points involved. The sampling was preformed on a machine with the minimal hardware requirement of Section 2.6, a Tablet PC with a Pentium 3 700 MHz processor and 256 MB of ram. Examined were 170 randomly drawn stroke objects which had intersection points computed and recorded.

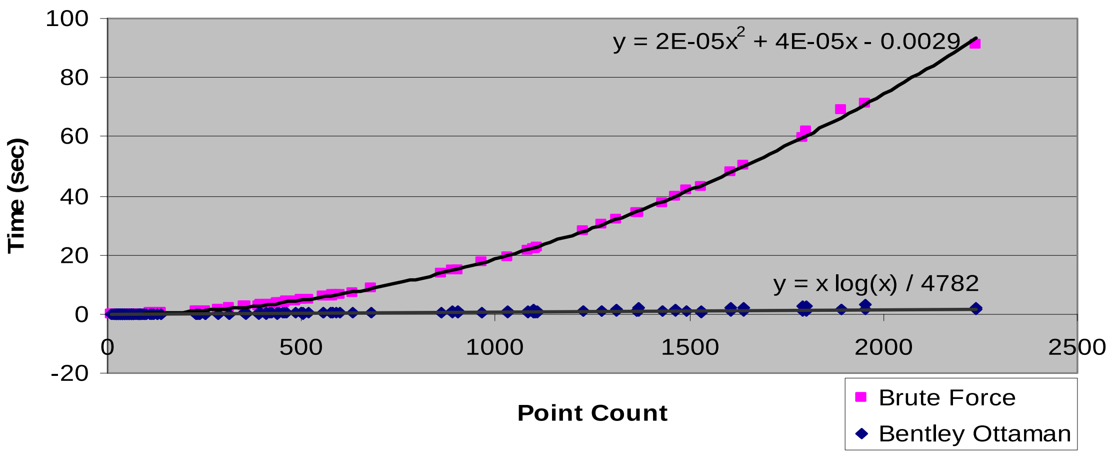


Figure 8-25 Overview of Brute Force compared to Bentley-Ottmann algorithm, illustrating the time required number of points in the Stroke object. The scale of this graph represents *Sets* of *Stroke* or *Strokes* objects, of the size to represent entire equations.

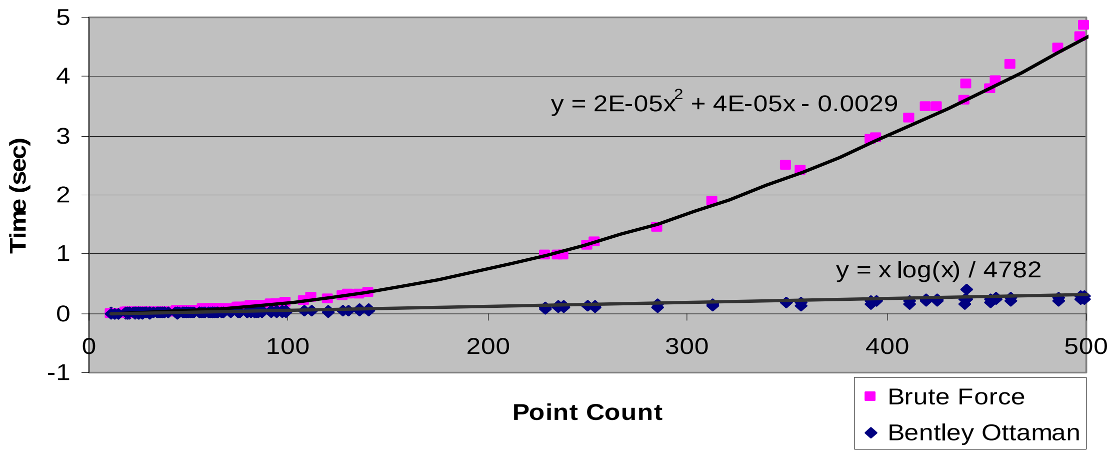


Figure 8-26 Overview of Brute Force compared to Bentley-Ottmann algorithm, illustrating the time required per number of points in the Stroke object. This scale of this graph represents expected size of *Stroke* objects, of the size to represent individual characters.

As seen in Figure 8-26, even for computing intersections in simple cases, the lower computational complexity of Bentley-Ottmann algorithm outweighs the simplicity of the brute force algorithm. In our implementation of nearly 2000 lines of code (excluding the standard template library (STL) objects *Set* and *Vector*), the Bentley-Ottmann algorithm will still be able to outperform the 10 line brute force algorithm.

Superimposed on top of the data points are approximate trend lines. In each figure, the upper parabolic line represents the brute force algorithm, and the lower almost logarithmic line represents the Bentley-Ottmann algorithm.

8.11 Conclusions

Each decision for the implementation of the different parts of this thesis required examination and impact analysis. Early decisions, such as programming languages and platforms initially targeted, would leave a lasting impression on our framework, influencing its uptake by future developers or researchers.

We have described in detail in this chapter the investigating of several problems including: choosing languages, targeting individual platforms, user interface experiments and determining self intersections. In each case we have presented the problem and our results. Together, these form the elements of a cross platform mathematical framework.

Chapter 9 Conclusions & Future Work

We have examined all the primary issues in defining a digital ink architecture that can support mathematics. We established the hardware and software requirements of a recognizer; software must be able to capture and act on ink related events while hardware must have an interactive screen as well as the processing power and memory to provide a real time inking environment. Beyond hardware and software requirements, a mathematical framework also has requirements to ensure its success. Identified are four requirements: platform independence, high-level ink manipulation, device API abstraction and resource abstraction which ensure our solution will accommodate as many targeted devices as possible while still providing a full suite of functionality and resources.

Given that no dominant handwriting platform exists which could be used by our mathematical framework, the creation of a Portable Digital Ink Architecture or PDIA was necessary to ensure applications that make use of our framework are abstracted from the details of manipulating ink. For instance, ink manipulation such as normalization methods or property retrieval will become a part of the PDIA. Taking advantage of existing infrastructure where possible, the described and implemented three-tiered PDIA provides a means of addressing each of the identified requirements (platform independence, high-level ink manipulation, device API abstraction and resource abstraction) of a mathematical framework. Initial versions of PDIA will support the desktop, Tablet PC and Pocket PC platforms.

By illustrating the differences between string based or textual languages and mathematical or visual languages, one is able to understand text based handwriting recognizers cannot be adapted to recognize mathematics. There are three primary properties of mathematical expressions that affect recognition: symbol identification, segmentation and context. Understanding these properties presents a challenge that must be addressed in order to provide a functional mathematical recognizer.

In creating an online mathematical recognizer, it is clear that significant dependencies will exist on the underlying ink architecture. We identify those dependencies to the extent that it is portable. Such an organization will improve both the developer and end user experience.

After surveying methods used to recognize mathematics, we decided upon a stage process of mathematical recognition: data collection and normalization, symbol recognition and thirdly, structural analysis. ORCCA's vision introduces a fourth stage: context analysis, a post recognition process commonly used in recognizing string languages. We believe adding context analysis capabilities will significantly improve our results in the same ways handwriting recognition has benefited from the additions of dictionary and grammar checks.

Adding to the value of our framework is the introduction of seven requirements for a user interface: interactivity, minimal restrictions on screen size input or output size, non-penalty for entering mathematics, persistant storage and networkability, computationally intelligent, and finally support for all mathematical symbols. The requirements will ensure users will have a complete end to end solution, allowing scenarios that permit users to take notes on a Pocket PC, transmit over a network to a Tablet PC or desktop, further refine and ultimately print finished works on paper or to a manuscript for electronic distribution.

The vision of this thesis is to provide a well engineered solution that is efficient and extendable. During implementation, each decision made required an examination of its impact. These choices, i.e. programming languages and initially supported platforms will leave a lasting impression on our mathematical framework, influencing its uptake by future developers or researchers. The result is a framework for mathematics and a PDIA that is functional, providing a foundation to other members of the ORCCA research lab and the mathematical community.

As PDIA is a foundational technology, providing other applications and solutions a means of targeting their goals quickly, i.e. allowing research on recognition to occur without having to concentrate on ink manipulation, there are no dependencies within our solution beyond needing a standards-compliant C++ compiler. This document does identify many requirements ranging from software and hardware to user interface and other features, but this is part of the definition of a math framework and is not considered a dependency.

With respect to the PDIA, the foundational classes including: *Point*, *Line*, *Rectangle* and *Stroke* class have been implemented and tested. However due to time restrictions, the higher level classes: strokes and ink are well defined and prototyped, although not fully implemented.

9.1 Future Work

Almost every product from the field of software has high aspirations. The scope of this thesis is no exception, only a small subset of a complete mathematical framework is discussed in great detail, with other topics being outlined and presented with requirements. We present below a short list of features that we feel is necessary for future works to address.

As opposed to providing a list of features that would be desirable to see in future versions of PDIA, we limit future work only to additional requirements we see as being necessary for the success of the framework presented by this thesis.

9.2 Automatic Creation of Strokes Objects

Groups of strokes or *Strokes* currently have to be added manually. Future work will need to provide considerable dedication to the automatic segmentation of ink strokes. A long outstanding problem, the ability to analyze and understand the physical layout of ink input has been addressed by substantial research including most recognizer prototypes.

9.3 Networking Capabilities

While the ability to support networking was considered in this thesis, there is no chapter dedicated to the topic. Beyond acknowledging that InkXML supports streaming ink packets, the impact of networking support have not been studied. While we see no major implications in adding network support to PDIA or as a requirement in general to the math framework, additional studies should be made to ensure the risks and effects are well understood.

References

[1] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. Proceedings of the IEEE, 80(7):1029-1058, July 1992

[2] C. C. Trappert, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(8):787-808, 1990

[3] R. Narasimhan. Labeling sSchemata and Syntactic Descriptions of Pictures. Information and Control, 7:151-179, 1964.

[4] A. Beláid and J.P. Haton. A Syntactic Approach for Handwritten Mathematical Formula Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 6(1):105-111, January 1984.

[5] L. H. Chen and P. Y. Yin. A System for On-Line Recognition of Handwritten Mathematical Expressions. Computer Processing of Chinese and Oriental Languages, 6(1):19-39, June 1992

[6] Y. A. Dimitriadis and J. L. Coronado. Towards an ART Based Mathematical Editor that uses On-Line Handwritten Symbol Recognition. Pattern Recognition, 29(6):807-822. 1995

[7] H. J. Lee and M. C. Lee. Understanding Mathematical Expressions Using Procedure-Oriented Transformation. Pattern Recognition, 29(3):447-457, 1994.

[8] De Solla Price, Derek J. "Ancient Greek Computer" Scientific American, June 1959, pages 60 - 67

[9] Bashe, Charles J.; Lyle R. Johnson; John H. Palmer; Emerson W. Pugh, IBM's Early Computers, MIT Press (1985).

[10] Ivan E. Sutherland. "Sketch pad a man-machine graphical communication system". Proceedings of the SHARE design automation workshop, January 1964, pages 329 - 346.

[11] xThink News: MathJournal Released. 28 July 2004. xThink Corporation.. http://www.xthink.com/company_news.html. 1 August 2004.

[12] Avitzur, Ron "Your Own Handprinting Recognition Engine", Dr. Dobbs Journal, April 1992

[13] Steve Smithies, Kevin Novins, James Arvo. A handwriting-based equation editor. In Proceedings of Graphics Interface '99, June 1999, pages 84 - 91.

[14] Nicholas E Matsakis. Recognition of Handwritten Mathematical Expressions. Master thesis, MIT, 1999.

[15] Richard Zanibbi. Recognition of mathematics notation via computer using baseline structure. Technical Report ISBN-0836-0227-2000-439.

[16] Sobel, Alan, "Electronic Numbers", Scientific American, June 1973, pages 64 - 73.

[17] Bager, J., and Bleich, J. (2000): WAP-Galerie. C'T Magazin Septemeber 2000, pages 200 - 207.

[18] Myers, B. A., Stiehl, H. and Gargiulo R. Collaborating Using Multiple PDAs Connected to a PC. Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'98), 1998, pages 285 - 294.

[19] Rob Jarrett, Philip Su. “Building Tablet PC Applications”. Microsoft Press, September 2002.

[20] Hideyuki Hayashi, Sheila Duncan and Susumu Kunto. “Computational Linguistics: Graphical Input / Output of Nonstandard Characters”. Communications of the ACM Volume 11 / Number 9 / September 1968, pages 613 – 618

[21] Moore, Gorden E. “Cramming More Components onto Integrated Circuits”. Electronics, April 19, 1965, pages 114 - 117.

[22] Hull, Jonathan. “A database for handwritten text recognition research”. IEEE Transactions Pattern Analysis and Machine Intellegince,vol. 16, no. 5, 1994, pages 550 – 554.

23 PalmSource Press Release: PalmSource Ships Faster, More Powerful Palm OS 5. PalmSource Inc. June 10, 2002. http://www.palmsource.com/press/2002/061002_1.html

[24] Microsoft Tablet PC SDK 1.7: http://msdn.microsoft.com/tabletpc . Microsoft Corporation. 2004

[25] Paul Thurrot. SuperSite for Windows: WinHEC 2004 Report. http://www.winsupersite.com/reviews/winhec_2004.asp . May 2004

[26] Carsten Magerkurth and Thorsten Prante. Towards a Unifying Approach to Mobile Computing. SIGGROUP Bulletin. Vol 22, No. 1. April 2001, pages 16 - 21.

[27] W3C Ink Markup Language (InkML). http://www.w3.org/2002/mmi/ink

[28] Todd Allen, Robert Nix, Alan Perlis. PEN: A Hierarchical Document Editor. Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation. 1981, pages 74 – 81.

[29] Xiaojie Wu. Achieve Interoperability of Pen Computing among Heterogeneous Devices and Digital Ink Formats, Master thesis, UWO 2004.

[30] W. Martin. Computer input/output of Mathematical Expressions. Proceedings of Second Symposium on Symbolic and Algebraic Manipulations. New York, 1971, pages 78 - 87.

[31] Dorothea Blostein. General diagram-recognition methodologies. Lecture Notes in Computer Science, Volume 1072, pages 106-122. Springer Verlage, New York. 1995.

[32] Dorothea Blostein and Ann Grbavec. Recognition of Mathematical Notation: Handbook on Optical Character Recognition and Document Image Analysis. World Scientific Publishing Company, 1996.

[33] Kam-Fai Chan, Dit-Yan Yeung. Mathematical Expression Recognition: A Survey. International Journal on Document Analysis and Recognition, Vol 3, No 1, 2000, pages 3-15.

[34] Kim Marriott, Bernd Meyer, and Kent D. Wittenburg. A Survey of Visual Languages Specification and Recognition. Visual Language Theory. Springer-Verlag, New York. 1998, pages 5 – 80.

[35] R.H. Anderson. Syntax-Directed Recognition of Hand-Printed Two-Dimensional Equations. PhD thesis, Harvard University, Cambridge, MA. January 1968.

[36] A. Beláid and J.-P. Haton. A Syntactic Approach for Handwritten Mathematical Formula Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 6(1):105-111, January 1984.

[37] K. F. Chan and D. Y. Yeung. Recognizing On-Line Handwritten Alphanumeric Characters through Flexible Structural Matching. Patter Recognition, 32(7):1099-1114, July 1999

[38] L. H. Chen and P. Y. Yin. A System for On-Line recognition of Handwritten mathematical expressions. Computer Processing of Chinese and Oriental Languages, 6(1):19-39, June 1992

[39] R. Fukuda, S. I, F. Tamari, M. Xie, and M. Suzuki. A technique of mathematical expression structure analysis for the handwriting input system. ICDAR'99, pages 131-134.

[40] Y. A. Dimitriadis and J. L. Coronado. Towards an ART based mathematical editor, that uses on-line handwritten symbol recognition. Pattern Recognition, 28 (6):807-822, 1995.

[41] M. Koschinski, H.-J. Winkler, and M. Lang. Segmentation and recognition of symbols within handwritten mathematical expressions. ICASSP'95, pages 2439-2442.

[42] S. Lehmberg, J.-J. Winkler, and M. Lang. A soft-decision approach for symbol segmentation within handwritten mathematical expressions. ICASSP'96, pages 3434-3437.

[43] H.-J. Winkler. HMM-based handwritten symbol recognition using on-line and off-line features. ICASSP'96, pages 3438-3441.

[44] H.-J. Winkler and M. Lang. Online symbol segmentation and recognition in handwritten mathematical expressions. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Volume 4, Munich, Germany, 1997, pages 3377-3380.

[45] Y. Sakamoto, M. Xie, R. Fukuda, and M. Suzuki. On-line recognition of handwriting mathematical expressions via network. ATCM'98 [4], pages 271-279.

[46] R. Marzinkewitsch. Operating computer algebra systems by handprinted input. Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation, Bonn, Germany, July 1991, pages 411-4134.

[47] Y. Nakayama. A prototype pen-input mathematical formula editor. Proceedings of ED-MEDIA 93 – World Conference on Educational Multimedia and Hypermedia, Orlando, FL, June 1993, pages 400 – 407.

[48] Bo Yu, Shijie Cai. A Domain-Independent System for Sketch Recognition. Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia, 2003, pages 141 – 147.

[49] R. Anderson. Two-dimensional Mathematical Notation. Syntactic Pattern Recognition, Applications, ed. K. S. Fu (Springer – Verlag, 1977), pages 147 - 177.

[50] S. Chang, A Method for the Structural Analysis of Two-dimensional Mathematical Expressions. Information Sciences 2, 3 (1970) pages 253 - 272

[51] A. Grbavec and D. Blostein. Mathematics Recognition using Graph Rewriting. Third International Conference on Document Analysis and Recognition. Montreal, Canada, Aug 1995, pages 417 – 421

[52] H. Twaakyondo and M. Okamoto. Structure Analysis and Recognition of Mathematical Expressions. Proceedings of the Third International Conference on Document Analysis and Recognition. Montreal, Canada. August 1995, pages 430 – 437.

[53] Vannevar Bush. "As We May Think". Atlantic Monthly, July 1945, Volume 176, No. 1, pages 101 - 108.

[54] Larry Press "Dynabook revisited—portable computers past, present and future". Communications of the ACM. Volume 35, Issue 3, March 1992, pages 25 - 32

[55] Ed Yourdon. "The Pen is Mightier than the Mouse". American Programmer, Volume 4, No 12, December 1991, pages 16 - 24.

[56] Leslie Lamport. LaTeX: A Document Preparation System (2nd Edition). Addison Wesley Professional. ISBN 0201529831. June, 1994.

[57] Hideyuki Hayashi, Sheila Duncan and Susumu Kunto. "Computational Linguistics: Graphical Input / Output of Nonstandard Characters". Communications of the ACM, Volume 11, Number 9, September 1968, pages 613 – 618.

[58] Todd Allen, Robert Nix, Alan Perlis. PEN: A Hierarchical Document Editor. Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation, 1981, pages: 74 – 81.

[59] L. H. Chen and P. Y. Yin. A System for On-Line Recognition of Handwritten Mathematical Expressions. Computer Processing of Chinese and Oriental Languages, 6(1):19-39, June 1992

[60] Gordon Kurtenbach and William Buxton. Issues in Combining Marking and Direct Manipulation Techniques. Proceedings of the 4th annual ACM symposium on User interface software and technology, Hilton Head, South Carolina, United States, November 1991, pages 137-144.

[61] Edward Lank, Jeb S. Thorley and Sean Jy-Shyang Chen. An Interactive System for Recognizing Hand Drawn UML Diagrams. Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative Research. Mississauga, Ontario, Canada, 2000, pages 7 – 22.

[62] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. Journal of the ACM. Volume 39, Issue 1, January 1992, pages: 1 – 54.

[63] Wolfgang Freiseisen, Petru Pau: A Generic Plane-Sweep for Intersecting Line Segments
RISC-Linz Report Series No. 98-18, November 1998.

Appendix A:

Copy of Survey Used to Collect Mathematical Handwriting Samples on the IBM CrossPad and Tablet PC Computers

Created by Dr. Stephen Watt, Ben Huang and Xiaojie Wu during 2002 – 2003, this survey was used to collect data on the IBM CrossPad. It later became the basis for the survey used to collect data on the Tablet PC.

☒ ☒

2 J Zka^e Zkb\ZdNpe [gdNZe hd^j

1.1 Letters and numbers

/	☐	0	☐	1	☐	2	☐
3	☐	4	☐	5	☐	6	☐
7	☐	8	☐				

]	☐	^	☐	_	☐	`	☐
a	☐	b	☐	c	☐	d	☐
e	☐	f	☐	g	☐	h	☐
i	☐	j	☐	k	☐	l	☐
m	☐	n	☐	o	☐	p	☐
q	☐	r	☐	s	☐	t	☐
u	☐	v	☐				

☒ Please click “Next Page” on the CrossPad when finish this page ☒

>		?		@		A	
B		C		D		E	
F		G		H		I	
J		K		L		M	
N		O		P		Q	
R		S		T		U	
V		W					

Please click "Next Page" on the CrossPad when finish this page

1.2 Greek letters

α		β		γ		δ	
ϵ		ε		ζ		η	
θ		ϑ		ι		κ	
λ		μ		ν		ξ	
k		π		ϖ		ρ	
σ		ς		τ		υ	
ϕ		φ		χ		ψ	
ω							

Γ		Δ		Θ		Λ	
Ξ		Π		Σ		Υ	
Φ		Ψ		Ω			

Please click “Next Page” on the CrossPad when finish this page

1.3 Calligraphic letters (only if *you* use them)

$\mathcal{A}$		$\mathcal{B}$		$\mathcal{C}$		$\mathcal{D}$	
$\mathcal{E}$		$\mathcal{F}$		$\mathcal{G}$		$\mathcal{H}$	
$\mathcal{I}$		$\mathcal{J}$		$\mathcal{K}$		$\mathcal{L}$	
$\mathcal{M}$		$\mathcal{N}$		$\mathcal{O}$		$\mathcal{P}$	
$\mathcal{Q}$		$\mathcal{R}$		$\mathcal{S}$		$\mathcal{T}$	
$\mathcal{U}$		$\mathcal{V}$		$\mathcal{W}$		$\mathcal{X}$	
$\mathcal{Y}$		$\mathcal{Z}$					

Please click “Next Page” on the CrossPad when finish this page

1.4 Script letters (only if *you* use them)

E		F		G		H	
I		J		K		L	
M		N		O		P	
Q		R		S		T	
U		V		W		X	
Y		Z		[		\	
]		^					

1.5 Open face letters (only if *you* use them)

E		F		G		H	
I		J		K		L	
M		N		O		P	
Q		R		S		T	
U		V		W		X	
Y		Z		[		\	
]		^					

1.6 Relations and their negations

$<$		$>$		$\leq$		$\geq$	
$\ll$		$\gg$					
$\subset$		$\supset$		$\subseteq$		$\supseteq$	
?		@		$\sqsubseteq$		$\sqsupseteq$	
$\sqcap$		$\sqcup$					
$\vdash$		$\dashv$		$\top$		$\bot$	
$\parallel$		$\mid$		$\in$		$\ni$	
;		$\sim$		$\approx$		$\simeq$	
$\overset{\sim}{;}$		$\equiv$		$\propto$			

$\nless$		$\ngtr$		$\nleq$		$\ngeq$	
$\not\subset$		$\not\supset$		$\nsubseteq$		$\nsupseteq$	
$\not\sqsubseteq$		$\not\sqsupseteq$		$\notin$		$\not\ni$	
$\not\mid$		$\not\equiv$		$\nsim$		$\not\simeq$	
$\not\approx$		$\not\doteq$		$\not\asymp$			

1.7 Arrows and pointers

$\leftarrow$		$\rightarrow$		$\uparrow$		$\downarrow$	
$\Leftarrow$		$\Rightarrow$		$\Uparrow$		$\Downarrow$	
$\leftrightarrow$		$\Leftrightarrow$					
$\hookleftarrow$		$\hookrightarrow$		$\rightleftharpoons$		>	
$\nearrow$		$\searrow$		$\swarrow$		$\nwarrow$	

Please click “Next Page” on the CrossPad when finish this page

1.8 Mathematical accents

a^3		a^{33}		$a\backslash$		az	
a[		$\bar{a}$		aY		a^z	
$\vec{a}$		$\mathring{a}$					

1.9 Other binary operators

$\pm$		$\mp$		$\times$		$\div$	
$\cap$		$\cup$		$\vee$		$\wedge$	
$\uplus$		$\oplus$		$\otimes$		$\odot$	
$\circ$		$\bigcirc$		$\cdot$		$*$	

Please click “Next Page” on the CrossPad when finish this page

1.10 Various other symbols

$\aleph$		£		$\wp$		4	
$\Re$		$\Im$		∂		∇	
P		R		Q		]	
$\surd$		$\emptyset$		$\forall$		$\exists$	
$\neg$		∞		_		$\pm$	
$\lfloor$		$\rfloor$		$\lceil$		$\rceil$	
.		-		(			

Please click “Next Page” on the CrossPad when finish this page

3 J Zka^e Zkb\ZdD ohi^jjbgf NZe hd^j

2.1 Fractions

$\frac{p\ ,\ q}{1}$

olfdqf enov

$0-\frac{y}{x}$

olfdqf enov

$$\frac{0}{a-\frac{b}{c-\frac{d}{e}}}$$

olfdqf enov

2.2 Roots

$\sqrt[q]{a\ ,\ b}$

olfdqf enov

$\mathrm{k}_{7}\,\overline{-q\ ,}\ \mathrm{j}\,\overline{q^4\ ,\ p^5}$

olfdqf enov

2.3 Sums and products, etc

$1\ \mathrm{P}^{k}_{f=3}\ a_f$

olfdqf enov

$\mathrm{P}_{\alpha^6 .\ \alpha .\ 3=2}\ \alpha\ \mathrm{hkc}x$

olfdqf enov

$\mathrm{P}\ {}^{k}_{f=2}\ *-0+^{k-f}a_f k^f$

olfdqf enov

$\mathrm{Q}^{k}_{f=3}\ *i\ ,\ \frac{0}{i}+$

olfdqf enov

$\mathrm{Q}\ {}^{k}_{h=3}\ *0\ ,\ a_h+\geq\ 0\ ,\ \mathrm{P}\ {}^{k}_{h=3}\ a_h$

olfdqf enov

$\mathrm{P}^{k}_{h=g}\ \mathrm{Q}^{j}_{g=3}\ f*j,k+$

olfdqf enov

$\mathrm{P}_{\substack{f\geq 3?g\leq 32 \\ f.\ g=34}}\ x^f y^g$

olfdqf enov

Please click “Next Page” on the CrossPad when finish this page

2.4 Limits and other operations

$\text{hei}_{\text{s\% 4}} \frac{\text{hj œj}\ \pi x}{\text{hj œj}\ x}$

olfdqf enov

$\text{hei}_{\text{k\% 4}} \overset{\text{P}}{\ } {}^{\text{k}}_{\text{f= 3}} \frac{0}{iz}$

olfdqf enov

$\underset{\text{q5 T}}{\text{T}} \phi\text{*}u\text{+} \underset{\text{f= 2}}{\text{\$}} \phi_{\text{f}}$

olfdqf enov

$\underset{\text{p5 S}}{\text{U}} P\text{*}t\text{+}$

olfdqf enov

$\underset{\text{f= 3}}{\forall} P_{\text{f}}$

olfdqf enov

$\underset{\text{lql= 3}}{\text{i]t}} |Mu|$

olfdqf enov

$\underset{\text{p5 ZZ?3 [}}{\text{i]t}} f\text{*}t\text{+}$

olfdqf enov

2.5 Derivatives

$\frac{\partial u}{\partial t}$; $h^4 \left(\frac{\partial^4 u}{\partial x^4} \text{ , } \frac{\partial^4 u}{\partial y^4} \text{ , } \frac{\partial^4 u}{\partial z^4} \right)$

olfdqf enov

$\frac{`}{`t}\phi$

olfdqf enov

$\nabla_{\mathrm{q}} f \star u, v+$

olfdqf enov

$\nabla \cdot u$

olfdqf enov

$\vec{\nabla} \times \vec{f}$

olfdqf enov

2.6 Integrals

$\mathrm{X}_{\mathrm{s}^6 \,.\, \mathrm{t}^6 \leq \mathrm{Q}^6} f *x, y+dx\,dy$

o lfdqf enov

$\mathrm{X}_{2}^{\mathrm{s}} e^{-\mathrm{p}^6}\,dt$

o lfdqf enov

$\mathrm{X}_{2}^{\mathrm{a}} dz \;\mathrm{XX}_{\mathrm{E}\,|} \frac{z}{c}\,dx\,dy$

o lfdqf enov

$\mathrm{K}_{\mathrm{D}} P\,dx \;,\; Q\,dy \;;\; /$

o lfdqf enov

$\}_{\mathrm{R}} f\,\text{`}s$

o lfdqf enov

$\mathrm{X}_{\mathrm{a}} \text{`}\omega \;;\; \mathrm{X}_{\mathrm{Ba}} \omega$

o lfdqf enov

$\mathrm{X}_{2}^{4} - \frac{0}{x}\,dx$

o lfdqf enov

Please click “Next Page” on the CrossPad when finish this page

2.7 Matrices and arrays

olfdqf enov

$$\begin{array}{l}
a_{33}x_3 \texttt{ , } a_{34}x_4 \texttt{ , } \cdots \texttt{ , } a_{3k}x_k \texttt{ ; } b_3 \\
a_{43}x_3 \texttt{ , } a_{44}x_4 \texttt{ , } \cdots \texttt{ , } a_{4k}x_k \texttt{ ; } b_4 \\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
a_{k3}x_3 \texttt{ , } a_{k4}x_4 \texttt{ , } \cdots \texttt{ , } a_{kk}x_k \texttt{ ; } b_k
\end{array}$$

olfdqf enov

$$\begin{array}{c}
\texttt{s}\_\_\_\overset{\texttt{k.3}}{\texttt{vu}}\_\_\_\texttt{t} \\
\begin{array}{cccccc}
4 & & & & & 5 \\
 & a_3 & / & \ddots & / & \\
\texttt{E} & a_4 & a_3 & \ddots & / & \texttt{F} \\
\texttt{E} & \vdots & a_4 & \ddots & \vdots & \texttt{F} \\
\texttt{E} & a_k & \vdots & \ddots & / & \texttt{F} \\
\texttt{E} & / & a_k & \ddots & a_3 & \texttt{F} \\
\texttt{E} & \vdots & / & \ddots & a_4 & \texttt{F} \\
\texttt{C} & \vdots & \vdots & \ddots & \vdots & \texttt{D} \\
 & / & / & \ddots & a_k &
\end{array}
\end{array}
\texttt{,k.3-×,k.3-}$$

olfdqf enov

$$\begin{array}{c}
\begin{array}{ccc}
4 & & 5 \\
\texttt{E} & \begin{vmatrix} x_{33} & x_{34} \\ x_{43} & x_{44} \end{vmatrix} & \texttt{F} \\
\texttt{C} & x & \texttt{D}
\end{array} \\
y
\end{array}$$

$$\sum_{l_5 @ l_6 @ \cdots @ l_{q-o}}^{,3?43 \ggg 3k-} \Delta^{l_5 l_6 \ggg l_{q-o}}_{l_5 l_6 \ggg l_{q-o}} \sum_{m_5 @ m_6 @ \cdots @ m_o} \begin{vmatrix} a_{m_5 m_5} & \cdots & a_{m_5 m_o} \\ a_{m_6 m_5} & \cdots & a_{m_6 m_o} \\ \vdots & \ddots & \vdots \\ a_{m_o m_5} & \cdots & a_{m_o m_o} \end{vmatrix}$$

olfdqf enov

2.8 Lines above and below formulas

$\overline{\overline{a}^4 \, , \, \underline{xy} \, , \, \overline{\overline{\overline{z}}}}$

olfdqf enov

$\{\overbrace{\overset{h\ ^4o}{a, \cdots, a}}, \overbrace{\overset{i\,'^4o}{b, \cdots, b}}\}$
$\underbrace{\qquad}_{h.\ icicj\ ckpo}$

olfdqf enov

2.9 Subscript and superscript

$b^{\frac{5}{6}}$

olfdqf enov

$f_{22\,23}$; $f_{23\,22}$

olfdqf enov

$Q^{\mathtt{fg}}$; $R^{\mathtt{fgh}}{}_{\mathtt{igh}}{}^{\mathtt{i}}$

olfdqf enov

$e_{\mathtt{,h-}}{}^{\mathtt{f}}\frac{\partial}{\partial x^{\mathtt{f}}}$; $\overset{\mathtt{,h-}}{\omega}{}_{\mathtt{g}}dx^{\mathtt{g}}$

olfdqf enov

$x_{\mathtt{fg}}y_{\mathtt{h}}$

olfdqf enov

$a^{\mathtt{s}}i$

olfdqf enov

$x^{\mathtt{t}|}$

olfdqf enov

Please click “Next Page” on the CrossPad when finish this page

2.10 Other expressions

cy , dx

olfdqf enov

p]j $\frac{x}{1}$; $\frac{\text{œj}\,x}{0\text{ , \_ko}x}$

olfdqf enov

$\|x$, $y\| \leq \|x\|$, $\|y\|$

olfdqf enov

$\|xy\|$; $\|x\| \cdot \|y\|$

olfdqf enov

$\frac{41\%}{02\%02\%15\%}$

olfdqf enov

$\binom{n}{m}$; $\frac{n\%}{m\%{*}n-m{+}\%}$

olfdqf enov

$\lfloor -x \rfloor$; $-\lceil , x \rceil$

olfdqf enov

Please click "Next Page" on the CrossPad when finish this page

$x \equiv y$, 0 *i k‘ m^4+

o lfdqf enov

$\{x$ 9$x >$ 4$\}$

o lfdqf enov

œj 07° ; $\frac{0}{3}$*$\sqrt{4}$ − 0+

o lfdqf enov

$\vec{x}$ $\overset{\text{abc}}{;}$ *$x_3, \cdots x_{\mathrm{k}}$+

o lfdqf enov

$A \xrightarrow{\alpha^4} B \xleftarrow{\beta^4} C$

o lfdqf enov

$g^\circ \mapsto g^\bullet$

o lfdqf enov

x 9; y

o lfdqf enov

Please click “Next Page” on the CrossPad when finish this page

$\mathtt{E} \otimes_{\mathtt{K}} \mathtt{G}$

olfdqf enov

$M\ \mathtt{;}\ {}^{\mathtt{M}}{}_{\mathtt{k5R}}\, M_{\mathtt{k}}$

olfdqf enov

$V_6 \setminus S_9/S_6 \oplus S_6$

olfdqf enov

$F\ \mathsf{9O}\ \times \mathsf{O}^{\mathtt{n}} \times \mathsf{O}^{,\mathtt{b.\ 3\text{-}k}} \longrightarrow \mathsf{O}^{\mathtt{j}}$

olfdqf enov

$\tau{*}G+\ \mathtt{;}\ \tau{*}G - e+,\ \tau{*}G \downarrow e+$

olfdqf enov

${*}G - e+^{*} \simeq G^{*} \downarrow e^{*}$

olfdqf enov

$L^{@\mathtt{k.\ 3A}}\ \mathtt{;}\ \mathtt{X}L^{@\mathtt{kA}}, L\,\mathtt{Z}$

olfdqf enov

Please click “Next Page” on the CrossPad when finish this page

$\sim A$+ ; $\overline{\sim A}$	olfdqf enov
0 $- x$; $=y$	olfdqf enov
1oyz	olfdqf enov

I irrevocably give permission for the ORCCA laboratory and its direction to use these handwriting samples for any purpose whatsoever without any fee.

Name: ______________________

Signed: ______________________

Date: ______________________

Witness: ______________________

Please click “Next Page” on the CrossPad when finish this page

Curriculum Vita

Name:	Kevin J. Durdle 2771 First Ave, Suite 305 Seattle, Washington, United States 98121
Post-secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada N6A 5B8 1999 – 2003 B.Sc
Related Work Experience :	Teaching Assistant The University of Western Ontario 2003 – 2004 Research Assistant Ontario Research Center for Computer Algebra The University of Western Ontario 2003 – 2004