MathBrush web application: Design and implementation of an online pen-input interface for computer algebra systems

by

Connor Flood

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Several pen-math systems have been developed for mobile and tablet platforms, most notably by the MathBrush project. With the increasing variety of available devices and platforms used by students, this thesis aims to design and implement a version of MathBrush for the web, such that it can be accessible from any device with a web browser. First, a formative study is conducted to gain a current understanding of the common processes used by post-secondary math students for completing assignments, such as: discussing the reliance of using paper, and identifying benefits/limitations of current tools used. Second, the MathBrush web application is implemented which requires creating a new architecture to support the web-based features of the application. Finally, a user study is performed to gain feedback from current math students. This feedback will highlight students' opinions on the application, and will relate back to discussions from the formative study to determine the overall usability of the MathBrush web application.

Acknowledgements

To begin, thank you very much to both of my supervisors: George Labahn and Ed Lank. Attending Waterloo has long been a dream of mine, and I am thrilled I got to experience graduate studies here under your supervision. Ed, our discussions in your HCI course were instrumental in providing inspiration for this thesis and very much appreciated. George, your continued support over the past two years has been highly motivating and rewarding. The flexibility you provided during this process was not only beneficial to the thesis, but also gave me time to gain teaching experience and apply for my dream job; both of which were significant in shaping my future ambitions.

Thank you to Mirette for all of your help and patience during the development process. There was no shortage of frustrating bugs along the way, but your knowledge and support was extremely valuable the entire time.

Thanks also to the additional readers: Dan Vogel and Arne Storjohann. Your time and input were well appreciated during the revision process.

Although sometimes it felt like it, this thesis did not consume all my time here in Waterloo. I was incredibly lucky to have shared some great memories with my awesome roommates, all of whom I consider great friends (but rivals at chel). Here's to the Glen Forrest boys: JJ, Adam, Ehhhic, Poopy, and Mouj. I originally typed out your real names, but that didn't seem right...

Lastly, an enormous thank you to my parents and brother back home in the Soo. I have never once doubted your support, and I feel very proud to make you proud. Your visits to Waterloo were always enjoyable, and I hope they continue in California. Can't wait to see what's to come!

Dedication

Dedicated to my extremely supportive grandparents. I'm so fortunate and appreciative to receive your continued love and encouragement. Love you all!

Grandpa (Alfred) Johns, Grandma (Jackie) Johns, Grandpa (Charlie) Flood, Grandma (Lynne) Flood, and Great Grandma (Elsie) Johns.

Table of Contents

List of Figures

Chapter 1

Introduction

The area of mathematics has been subject to a revolution in both how material is taught, as well as the methods students use for studying and completing assignments. A large part of this change is due to the introduction of computer algebra systems (CAS). When used properly, CAS offer the ability to eliminate much of the "manual labour" associated with performing computation-based math problems. As a result, it is not a surprise that such systems would appeal to math students. The combination of the Internet's accessibility along with the functionality of CAS seem like an ideal tool for math students, especially at the university level when more difficult computational topics are introduced.

The motion towards an increased usage of CAS can seemingly imply a smaller dependence on using pencil/paper, a tool which mathematicians have traditionally preferred. While there exist a great number of advantages to this movement, there also exist a number of notable limitations which will be examined in this thesis. One large limitation being the typical requirement of learning the syntax of a computer algebra system in order to properly interact with it. For a student, this may not be ideal when they would rather simply input problems by drawing proper math notation, similar to how they used pencil/paper. This introduces the concept of pen recognition, whose implementation can present its own set of challenges.

Given that the technology supporting web applications, pen recognition, and CAS have been around independently for many years now, there exist several well-known products (Maple, Wolfram|Alpha, MathBrush, etc) which attempt to combine a subset of these technologies. These products offer terrific services to support work completed by math students in their courses. Despite these developments, there does not appear to be an available product to students which offer the benefits of combining pen recognition as a

method of input to computer algebra systems, in the form of a web application.

This thesis determines to evaluate the extent to which an online pen math system can play an important role in assisting students in their math courses. The implementation of this application will be an extension of MathBrush, an existing pen-based system developed specifically for the iPad and Microsoft devices (i.e. Surface Pro). By redeveloping MathBrush as a web application, MathBrush is no longer dependent on any particular piece of hardware or operating system, and can be run on the web while still maintaining the advantages of interacting with CAS (such as Maple and Sage) via pen-based input. The success of this development can hopefully prove to be a useful tool for students, and act as a milestone for the accessibility and usability of computer algebra systems.

1.1 Computer algebra systems (CAS)

Computer algebra systems (CAS) have been widely used to support the teaching and learning of mathematics at the university level [34]. In addition to having the capability of manipulating the algebraic symbolic form of equations, CAS are often used for the symbolic computation surrounding differentiation, integration, simplification, and solving differential equations [34, 25].

In the past, many of the limitations surrounding CAS were interface related [35]. Examples of these limitations included: the use of unnatural linear notation to enter and edit expressions, displaying large expressions that run off the screen, and complicated command-based methods to select subexpressions. These user interface problems often intimidate novice users and cause frustration with experienced users. A common notion has been that users are more likely to take advantage of CAS for its ability to perform tedious computations, so long as the interface allows for natural and intuitive input. The benchmark for creating such an interface is that it would ideally correspond to typical pencil and paper manipulations [35].

Today's research and development of CAS is mainly driven by three goals: wide functionality, speed, and ease of use (i.e. user interface and graphics display) [36]. This thesis will concentrate mostly on the latter goal. Three of today's most widely CAS, which will be referenced in this thesis are: Maple, Sage, and Mathematica (accessed through Wolfram|Alpha). While the underlying goals of these CAS are similar, each system offers their own unique benefits and sets of features in order to create a suitable environment for students and researchers.

1.1.1 Maple

The Maple CAS is a technical computing and documentation desktop environment created by researchers at the University of Waterloo in the early 1980s [30, 31], and is now owned by Maplesoft [12]. The syntax for Maple's dynamically typed programming language goes by the same name. The traditional (and still widely common) approach for interacting with the Maple CAS is performed by entering queries in a command-line interface (CLI), or by writing and executing scripts, both of which are written in the Maple language. Over time, Maplesoft has continued to enhance their product in terms of speed, range of functionality, and in its interface. With university students being a primary target audience, Maple has added user interface features that decrease the reliance on a student's knowledge of the Maple syntax. Maple's "Clickable Calculus" allows novice users to perform complex operations without knowing any special commands or syntax [12]. In terms of accessibility, Maplesoft now also offers the product "MapleCloud", which allows access to the CAS engine from any PC or mobile device with an Internet connection. While the Maple desktop environment requires a software license, MapleCloud is accessed from a device's web browser and does not require a Maple installation or license on the device.

1.1.2 Sage

Sage (a.k.a. Sagemath) is a free open-source mathematics software system licensed under the GPL [13]. Sage builds on top of various open-source packages, including: Sympy, R, matplotlib, NumPy, etc. Interacting with the Sage CAS can be accomplished by two different methods. The first method is by using the Sage "Read-Eval-Print-Loop" (REPL), based on IPython. The REPL is a command-line interface for inputting mathematical queries using the Sage syntax. The other method of input makes use of the Sage Notebook, which allows a user to write a program with Sage code (i.e. series of Sage statements). Neither of these options provide the user with a graphical user interface (GUI). This means that a knowledge of the Sage syntax is required in order to properly use the Sage CAS. Due to the open-source nature of Sage, it is free for anyone to use and can easily be downloaded online. Users are able to interact with Sage locally on their desktop, as well as through the cloud implementation of SageMath in a web browser.

1.1.3 Mathematica

The Mathematica CAS was first introduced by Wolfram Research in 1988, and is still today Wolfram's flagship product [14]. The language used for interacting with Mathematica is

called the "Wolfram Language" (recently changed from just being called "Mathematica"). Similar to Maple and Sage, Mathematica offers notebooks as a means of writing programs to be computed by the CAS (indeed Mathematica was the first to use a notebook interface on Mac and NeXT machines). Also similar to Maple, Mathematica requires a license on desktop machines in order to run computations with the CAS locally. Mathematica has also developed a cloud system ("Wolfram Cloud") which allows users to create and run Mathematica programs inside a web browser without requiring any local license or installation. Much of the Mathematica kernel development has been dedicated to improved efficiency and a broader range of functions. In terms of the user interface, there has not been much change to the Mathematica Notebook since its inception. Interacting with the notebook heavily relies on the user's knowledge of the Wolfram Language. However, the web application "Wolfram|Alpha" has become a popular tool for students as a means of communicating with the Mathematica CAS.

Wolfram|Alpha

Wolfram|Alpha is a computational knowledge engine, which accepts a string of natural language as its input. Introduced in 2009, Wolfram|Alpha provides an interface very similar to Google which allows a user to type a query into a toolbar in plain English. Natural language processing (NLP) is then used to determine what Wolfram|Alpha believes the user is asking for. Instead of then displaying a list of links to relevant information (like Google's approach), Wolfram|Alpha attempts to directly display whatever information the user may have been asking for. For example, if a user inputs into Wolfram|Alpha "what is the integral of 7x", Wolfram|Alpha will return a result pod which displays the computed result of $7x^2/2$. Wolfram|Alpha also offers step-by-step solutions (a paid feature) for many of its results. In addition to accessing Wolfram|Alpha from the web application, users can also access it via an Android/iOS application, as well as via the "WolframAlpha" function which exists in Mathematica.

1.2 CAS use in education

Given the range of benefits that CAS offer to mathematicians and researchers, it seems logical that CAS would also provide redeeming benefits in math education. First year math courses (e.g. Calculus I and II) are often heavily computation based, and thus are typically composed of topics well-covered by CAS. Computer algebra systems are gradually being introduced into an increasing number of university math courses in an effort to make

the teaching and learning process more meaningful [15]. One particular study proposed that students "mastering" Maple during their first few years at university is beneficial not only in solving mathematical problems, but also in any complex applied problems they may encounter in the future [21]. Specifically in lab exercises, students "learn by doing it" when it comes to applying math concepts taught in class towards interacting with a CAS.

While several previous studies [15, 21, 34] have examined the benefits of integrating CAS into math curricula, other studies have also noted some obstacles. An international survey by Lavicza argued that the major hindrance to integrating CAS in the classroom is the complexity of the CAS syntax [24]. Buteau et al. found in their qualitative study that syntax is the second most frequent concern for students and practitioners when it comes to using CAS in their math courses [17]. The largest issue identified for CAS integration in the same study was regarding assessment. Since the role of CAS is often to perform computations that were once done by hand, course assessments had to be modified in order to ensure certain learning objectives were still met by the students.

The integration of CAS in university math courses provide students a unique way of applying skills taught in the classroom. Previous work in examining the role of CAS in an education setting has not only verified this claim, but has also found certain areas of weakness or caution with this integration. A main area of concern seems to surround the syntax for interacting with CAS. Maple, Sage, and Mathematica all use their own unique languages for accepting inputs (in the form of a single statement or a script/notebook). One of Maple's solutions to this concern is in the form of the Clickable Calculus feature, which allows a user to construct an integral, for example, and does not require a working knowledge of the Maple syntax. Wolfram|Alpha allows users to type their input into a search bar using plain English. The extent to which these solutions succeed from a student's perspective in removing a rigid syntax will be evaluated in this thesis. The argument can be made however, that learning the proper syntax for interacting with CAS can become a valuable skill. Students who plan to conduct research in a math-related field will likely be required to use CAS, and would thus benefit themselves to already be comfortable with using proper syntax. Using the CAS syntax also removes any ambiguity or reliance on the CAS user interface. User interface features which remove the requirement of knowing the CAS syntax often only offer a small subset of features which can be accessed with a solid understanding of the syntax.

The other main area of concern identified in previous work was related to evaluation. If students have access to CAS when completing various assessments in their math courses, it does not make much sense for them to be tested on computation-based problems (since the CAS will be able to easily do that). It appears then that some adaption must take place in the course's design, specifically the types of assessments used, for students to be

tested properly if CAS are integrated into math courses. Asking an increased number of application-based questions may be a plausible solution, since it would be more difficult for CAS to provide direct answers for application questions, as opposed to computation questions. Another area of concern related to evaluation is academic dishonesty. When CAS are able to provide the final answers to computation questions, and even the step-by-step solutions in the case of Wolfram|Alpha, this can give cause for concern on the degree that a student is relying on CAS to complete assessments. It is nearly impossible to enforce how much a student uses CAS when completing an assignment, and thus can be difficult to distinguish between students who are able to perform computations on their own versus students who fully rely on CAS.

1.3 Existing technology

1.3.1 Pen recognition

One of the large limiting factors of CAS noted in previous studies relates to the form of user input. Users traditionally interact with these CAS by typing commands or scripts using the strict syntax of the CAS. While using syntax does present some key benefits (lack of transparency, exact interpretation, basic user interface, etc.), for students these benefits may often times be outweighed by some notable limitations, specifically having to learn the syntax of the language. Given that students are typically most comfortable with doing math using pencil/paper, it would be ideal for CAS to offer a similar form of input in order to make the transition from pencil/paper to CAS much smoother and more enjoyable.

An important distinction to make is the difference between optical character recognition (OCR) and sketch-based input. OCR is a topic surrounding the recognition of printed characters. This implies that the text (whether characters or mathematics) is often acquired through scanned images, which are then processed in an attempt to recognize the content of the scanned image [10]. Sketch-based input on the other hand, involves a user drawing their input directly into an application. This allows the application to have direct access to the ink (i.e. list of points) that the user has drawn, and thus eliminates the scanning stage. When considering mathematics specifically, it has been previously noted that recognizing mathematics in typset documents is significantly easier than in arbitrary handwritten documents [33]. This is due to typset mathematics having a clear and precise representation, whereas handwritten documents are typically subject to noise and other forms of inaccuracies.

There have been several pen-based (i.e. sketch-based) applications implemented with the intention of recognizing handwritten mathematics. These include [33]:

- MyScript products by Vision Objects: MathPad, Calculator, and Web Equation
- MathPad2 from Brown University (unrelated to MyScript's MathPad)
- Infty Project's InftyEditor handwriting interface
- S Note on Samsung's Galaxy Note (Android based)
- MathBrush

A table of features is shown below which compares each of these applications. The features that are being considered are:

- Handwritten recognition: Does the application recognize handwritten mathematics?
- Inplace recognition: Does the recognized expression replace the original ink? As opposed to providing separate views for the ink and recognition
- Correction: Does the application provide the ability to correct the recognition?
- Computations: Does the application compute or provide mathematical functions to perform on the recognition? And if so, which CAS is used?
- Multiple expressions: Does the recognition handle multiple expressions? E.g. matrices, systems of linear equations, etc.
- Platform: What platform(s) is the application available on?

	MyScript	MathPad2	Infty	S Note	MathBrush
Handwritten recog.	Yes	Yes	Yes	Yes	Yes
Inplace recog.	Yes	No	No	No	No
Correction	No	Yes	No	No	Yes
Computations	W|A	Generates graphs	No	No	Sage, Maple
Multiple expr.	Yes	Yes	No	No	Yes
Platform	Android, iOS, Windows, web, APIs	Desktop	Desktop	Android	iOS, Desktop

1.4 MathBrush

The MathBrush project is a standalone application designed and implemented for tablet PCs (Surface Pro) as well as iOS devices (iPad) [5]. The main focus of the project is to experiment and enhance the recognition of hand written mathematics, as well as investigating various user interface challenges. MathBrush recognizes mathematical expressions as you draw them, and recognizes a wide range of mathematics (including polynomials, integrals, and matrices) [22, 23]. The PC application offers a connection to the Maple and Mathematica CAS. This provides users the ability to perform various computations on their recognized expressions using these CAS. The iOS application has a very similar set of features, but uses Sage as the computer algebra system.

The MathBrush recognizer is identical for both the iOS and Windows platforms. A grammar based parser provides multiple interpretations of the input expressions. The recognizer and corresponding user interface for each platform also allow for the user to correct single characters, sub-expressions, or complete expressions. Previous works [26, 27, 28, 29, 33] describe the MathBrush recognizer in more detail. The majority of this thesis treated the recognizer as a "black box", and did not aim to directly modify or improve the recognizer.

1.4.1 MathBrush use cases

The integration of CAS in educational settings has been examined, with its main limitation appearing to be related to students learning syntax. The pen recognition features of MathBrush address this limitation by allowing for a syntax-free interaction with CAS. As a result, students are considered to be target users of MathBrush as the application could be beneficial when completing course requirements such as assignments. This thesis focuses mainly on use cases related to students using MathBrush during their math courses.

A study by Bunt et al examined the role of CAS in the area of mathematics research [16]. Specifically, pen math systems (such as MathBrush) were evaluated to determine the extent to which math researchers would find such tools useful during their typical workflow. Counter to their expectations, the study found that computational tools play only a minor role in the examined researchers' workflow. Much of this was due to CAS lacking in the following areas: transparency in explaining computed results, formality in CAS' input/output style dialogue, in-place manipulation of mathematical objects, and collaboration with other researchers. While this thesis is more focused on students as the end-users, these mentioned

limitations of CAS in research still provide important limitations to consider during the design and development of new tools.

1.5 Web application

The main focus of this thesis is investigating the reimplementation of MathBrush as a web application. One of the primary motivations behind this is extending the accessibility of the application. Since MathBrush was developed specifically for the iPad and Microsoft tablets, the application is also limited to running exclusively on these devices. Products like the Samsung Galaxy tablet (Android based) or any other devices running a Linux operating system will not be able to currently support the application. This can be problematic for two main reasons:

1. Users must own a compatible device in order to access the MathBrush application.
2. Developing native applications for various platforms (i.e. Windows, iOS) requires developing and managing separate code bases for each of these platforms. While certain libraries (i.e. the recognizer) may be reused for each platform, the user interface and front-end aspects of the applications must be developed specifically for each platform. As a result, any change to the interface in one codebase will likely require a change to the other managed codebases.

Developing a web version of MathBrush aims to solve the above two limitations. Users will be able to access the application from any device/platform which contains a web browser. This also allows for a single codebase to manage the web application. A side effect of developing a web version of MathBrush is that the application will also be accessible from non-touch screen devices. Desktop machines will be able to use the application, even without making use of touch-screen hardware. As a result, the application will be developed to also support a mouse click-and-drag form of input.

By removing the dependency on specific devices and platforms, a web version of MathBrush shows promise in the accessibility, maintainability, and longevity of the application.

1.6 Challenges

In terms of the development of a MathBrush web version, there were a few known challenges that needed to be faced. These challenges are summarized below (and will be discussed in

more detail in Chapter 3):

1. Pen input: The development environments for both the iOS and Windows versions of MathBrush contain corresponding frameworks which offer built-in libraries made specifically for pen/ink input. This includes the ability for ink, and individual strokes, to be drawn, recorded, deleted, and manipulated. As for web development (using a JavaScript approach), there are no built-in libraries for ink input. This requires each of these features associated with pen input to be implemented manually.

2. Calling the recognizer: The recognizer used by both MathBrush platforms is in the form of a compiled C++ dynamic linked library (DLL). The recognizer is run directly on the client, and thus does not require connections to any remote servers. From a web perspective, calling client-based C++ DLLs is not very common. Other languages like Java and Python have their own web frameworks which makes it easy to call functions/libraries in those languages directly from JavaScript. While certainly possible, determining how the web client (JavaScript) called a C++ DLL still presented a challenge. In addition, there did not appear to be any viable way in which the C++ DLL can be located on the web client. As a result, the recognizer instead acts as a server functionality.

3. Incompatible rending library: A core feature of MathBrush is not just the ability to recognize mathematical expressions, but also the ability to modify the recognized expressions. Creating a "perfect recognizer" is a near-impossible feat, and thus users of MathBrush should have the ability to correct the recognition when necessary. Unfortunately, the rendering library in MathBrush created for this purpose is not compatible with the JavaScript web implementation. Since this is not a challenge this thesis was able to overcome, user interface changes were made to modify the ink input (delete strokes, undo, etc) in the event an invalid recognition is made.

1.7 Outline

The overall structure of this thesis is outlined by three main parts (represented by the three subsequent chapters). The first part delves into the area of human-computer interaction (HCI), where the goal was to perform an analysis on the tools used when students are completing math courses. More specifically, examining the strengths/limitations of various tools and resources (i.e. pencil/paper, Maple, Wolfram|Alpha) that are often relied on for math assignments. This formative study is in the form of several individual semi-structured

interviews with students who have recently taken a university math course. The purpose of this stage is to provide justification and insight for the second stage: developing the MathBrush web application. This second stage is focused solely on the implementation of the web application, which will range from design considerations, the architecture of the application, implementation challenges, application limitations, and overall use cases. Results from the first stage are beneficial in ensuring that important use cases are met with the MathBrush web application, and that limitations discussed with other tools are avoided when possible. The final stage of this thesis is in the form of a user study, where users of the same demographic as the first stage (students who recently have taken a university math course) were asked to use the MathBrush web application. Participants were given certain math problems to solve using the application, and were also allowed to freely "play" with the application. This was followed by surveying the participants on what they viewed the strengths/limitations of the application were, and gaining feedback from their experience using the application. The results of the user study are then compared to the results from the formative study in order to examine the extent to which the MathBrush web application met certain limitations, and provide insight into whether the application can be considered a usable and useful tool for math students.

Chapter 2

Formative study

While much of this thesis is composed of the development and implementation of the MathBrush web application, there are several design and human-computer interaction (HCI) considerations to be made. The primary target user of the MathBrush web application is students enrolled in math courses, including introductory university math courses. In order to build software to reach this audience, it is beneficial to understand the software needs of students, based on tools they already use as part of their courses. This chapter aims to acquire information directly from students regarding expectations and desired features of math software they would find helpful when completing course requirements such as assignments. Understanding the individual processes that students take when completing assignments provides insight into what improvements can be made to benefit students in this process. Exploring the extent to which CAS have already been incorporated into students' math courses is examined. This includes determining under what conditions a student uses CAS, as well as what they view as strengths/limitations when using certain CAS, such as Maple or Wolfram|Alpha. Topics of discussion with students relating to the usability of a desired math application, such as the migration from using pencil and paper to pen recognition, are constructive in better designing the interface of the MathBrush web application. The interface is a reflection of the use cases outlined by students, ensuring that relevant and important features and ideas are supported by the web application.

This formative study was conducted with the web version of MathBrush in mind, and so much of the discussions were related to acknowledging or justifying the features of the web application. While a large amount of these discussions are fairly general, and could still be applied to the iOS version of MathBrush, the chosen topics of discussion were intended to support the design and features of the MathBrush web application.

2.1 Purpose

While many studies have shown how CAS can be integrated into high-school and university math curriculums, little recent work in the HCI community has examined the extent to which students voluntarily use CAS as part of their process for completing math assignments. This study seeks to determine the common methods that students follow when completing their math assignments, and the tools that are used alongside pencil/paper. It can sometimes be the opinion of students that "real mathematics" is done by hand [32]. This view likely arises from the traditional approach to completing math assignments, which is done with pencil/paper.

Examining the conditions when a student finds it beneficial to use CAS versus doing math "by hand" helps develop more specific use cases for students making use of math software, such as MathBrush. In addition, exploring strengths and weaknesses of commonly used math software is useful in understanding key features of CAS from the students perspective. It is the end goal of this study that particular themes arise from these discussions that will constitute the need for designing and developing improved math software to meet the needs of students in undergraduate math courses. Understanding how these students complete assignments, and how current computational tools integrate with their practices are necessary to understand the specific HCI challenges for this class of software. Many of the design considerations used for constructing the interface of the MathBrush web application stem from these discussions with students.

2.2 Related work

A large motivation of this study was to perform a replication study similar to that of Bunt, Terry, and Lank which examined CAS use in mathematics research [16]. The main difference between these studies is that in this study CAS use was examined in an education environment, specifically undergraduate mathematics. The methods for acquiring data however, are very similar, as discussed below.

2.3 Method

Data was acquired by reporting the findings from a qualitative study involving individual semi-structured interviews with 17 current university undergraduate students. All participating students have taken an undergraduate math course within the last four years.

The degree programs which each participant was enrolled in varied. The time constraint allowed for discussions to be focused on current and up-to-date tools and practices used as part of their math courses. The interview data provided meaningful themes, implications, and topics of discussion that were considered when assessing and designing math software geared towards students.

Data was collected by audio taping the interviews, which was immediately followed by the interviewer creating detailed notes on all topics that were discussed. Affinity diagrams were helpful in analyzing the data acquired from the participants' interviews. This was especially beneficial in extracting common themes, and making direct comparisons between the various tools discussed.

2.3.1 Semi-structured interviews

A set of questions and topics were prepared before the interviews to discuss with each participant. Maintaining the notion of "semi-structured" interviews, the interviews more so took the form of a conversation, rather than a strict question/answer format. Below are some of the major questions/topics that were discussed with all participants:

- "Walk me through the typical process of completing your math assignments"
- "What tools were used when completing assignments, and for what purposes?"
- Benefits/limitations of any tools used
- Reliance on using pencil/paper
- Contexts pen recognition could be beneficial
- Openness to changing how assignments are completed
- Suggestions on features of ideal math software

2.4 Findings

Various reoccurring themes were extracted from each of the semi-structured interviews, related to completing assignments, CAS usage, and pen recognition. The analysis of these findings was beneficial towards forming overall conclusions and justifying features and qualities implemented in the MathBrush web application.

2.4.1 Assignment completion process

Understanding the process for how students approach and complete assignments is critical to properly analyzing the extent to which various tools, such as CAS, meet their needs. Ideally, the primary goal of a student is not necessarily to just complete the assignment, but also to understand the concepts and topics that the assignment is composed of. It was found that the majority of participants followed a very similar process when completing assignments, with some minor differences. The following are the four main stages that were described by the participants for how they approach assignments:

1. Review course material: Read over course notes, and ensure a sufficient understanding of the material is grasped before attempting assignment questions.
2. Rough work: Attempt assignments questions independently, without relying on other tools or resources (CAS, online sites, etc.).
3. Verification: Compare attempted answers with solutions generated by CAS (Wolfram|Alpha, online calculators) and comparing with friends.
4. Final draft: Modify the initial solutions based on the verification stage. This often involves creating a new "clean" copy of the solutions.

When discussing the roles that CAS or other pieces of software play during this process for completing assignments, all participants indicated that software is used mostly for the verification stage. After attempting the assignment by hand using pencil/paper, participants would then typically double-check their solutions with online software systems (such as Wolfram|Alpha). More than half of the participants stated that on occasion they will also make use of this software when they get stuck on a question. Three of the participants claimed in some cases to make use of this software before even attempting the solution on their own.

2.4.2 Tools used

The three main tools that participants claimed to use throughout the process of completing assignments were pencil/paper, Wolfram|Alpha, and Maple. Preliminary readings suggested that CAS (in this case Wolfram|Alpha and Maple) would be beneficial to students in terms of experimenting and exploring with various mathematical concepts. While

this may be true in a classroom environment, discussions with the participants were helpful in examining their individual use of CAS to complete assignments on their own time. Pencil/paper was found to still play a major role in doing math, regardless of the numerous CAS and associated features that are available for students.

The heavy use of pencil/paper was not much of a surprise, nor were the benefits for doing so. Given its simplicity and ease of use, the participants explained pencil/paper played a key role when completing assignments, especially in the rough work and final draft stages. One main disadvantage of using pencil/paper was that students would still need to verify their answers with a CAS, most notably Wolfram|Alpha. Another point made was that if a change is made during a mathematical solution on paper, then there could be several subsequent changes that will also need to be manually changed.

It was interesting to find that all of the participants relied on Wolfram|Alpha in some form when completing their assignments. The majority of participants claimed to have used Wolfram|Alpha mainly for verification purposes. Others pointed out the benefits of some of its other features, such as step-by-step solutions and graph visualizations. Much of the discussions regarding Wolfram|Alpha were centralized around the input. "One of its strengths was that it could handle just about anything you could throw at it" (P5). Other participants pointed out the benefit to using natural language input. "It was nice being able to simply type integral of some function" (P1). The interpretation of the input was also viewed as an advantage to using Wolfram|Alpha. "It [Wolfram|Alpha] would tell you explicitly how it was interpreting your input" (P5). However, there were also several disadvantages regarding the input for Wolfram|Alpha. "Having to type in queries was sometimes cumbersome" (P4). Typing seemed to be a problem with several of the participants, especially as the complexity of the input increased. "When you have really long integrals it is hard to type in a single line" (P3). A point was also made that the input interpretation is not displayed until the results are shown. "I had to repeatedly check if I was typing the input in properly." (P1). One other limitation discussed was ambiguity in the input. "You could indicate the transpose by raising something to the power of a capital T, but if that variable already existed it would become ambiguous" (P5). While it is clear there are some concerns regarding the input limitations of Wolfram|Alpha, those concerns do not appear to outweigh the benefits of this CAS, given that nearly all participants voluntarily make consistent use of it.

Three of the participants (1, 3, 5) indicated that they were required to use Maple as part of the assignment completion process for their calculus courses. Advantages such as precision and accuracy were briefly discussed, but that was mainly directed towards CAS in general, and not specific to Maple. The most prominent concerns discussed regarding Maple seemed to be its apparent intimidating interface. "It was insanely overwhelming,

especially at a 1st year level. I didnt need 90% of the menus, I just needed the most basic stuff." (P1). In addition to viewing the large number of unused features as a disadvantage, the issue with learning syntax was also brought up. "You dont want to look up syntax just to check your answer, and then still maybe get it wrong" (P3). Besides focusing on the user interface and input mechanisms, Participant 5 discussed limitations in accessing the CAS. "If I wanted to use Maple, I had to go on one of the school computers, and I like to work from home" (P5). Since Maple is a desktop application which requires a site license, having to be on campus to use it is a fairly large limitation, especially when compared to a web/mobile application like Wolfram|Alpha.

2.4.3 Academic integrity

Academic integrity was not an expected topic of discussion to have had during the interviews with participants. It was first brought up as an immediate concern by Participant 3 when describing some of the features offered by CAS, more specifically using the step-by-step solutions from Wolfram|Alpha. "You are paying for the privilege of cheating" (P3). This topic was brought up again near the end of the interview, in which the same participant stated: "I don't think software that does your work for you has a place in assignments" (P3). Reasoning given behind this thought was that these resources are not available when writing the final exams. This raises an interesting point of questioning where to draw the line between beneficial features in CAS and it becoming academically dishonest. For notable CAS such as Maple and Wolfram|Alpha, it is not their goal to provide solutions to students so that they can be directly copied onto assignments without understanding the material. However, it is no surprise that some students will abuse the privilege of having access to CAS features, and use it in ways that could be considered academically dishonest. Being aware of this, should that warrant limiting features that CAS offer, such as step-by-step solutions, to uphold a certain level of academic integrity? Keeping in mind that a large motivation of CAS is to allow students to experiment and explore with mathematical concepts, it would be concerning if the students who were doing exactly this were now limited due to these same features being used for academically dishonest purposes. Participant 4 was asked to provide an opinion on this discussion, specifically related to whether step-by-step solutions should be removed to help prevent academic dishonesty. "If people are going to cheat, they're going to find a way anyway. Wolfram|Alpha might make that easier, but other sites will do the same thing" (P4). This follows along the notion that if cheaters are always going to find a way to be academically dishonest, then it does not make much sense to limit features of CAS for the students that are actually using features as they were intended. Participant 14 also made the point that

"looking online for answers will not help you for tests. Assignments were only worth 1% each, so you lose out more on the tests than the assignments" (P14).

2.4.4 Pen recognition

It was made clear that an advantage of using pencil/paper, and a big disadvantage when using CAS, is how input is formatted. In Maple, a strict syntax must be learned and followed in order to be used properly. An exception to this is Maple's "Clickable Calculus" feature, which promotes a heavier usage on the user interface in order to input queries, as opposed to following the syntax. For Wolfram|Alpha, a natural language input was well appreciated by the participants, but there still existed large limitations, specifically with the input interface. Participants were asked how they felt about using pen recognition, and in what scenarios it could be beneficial. "Using a stylus with Wolfram|Alpha and seeing the expressions being formed as you write would be really helpful" (P1). The majority of participants claimed that being able to incorporate pen recognition into Wolfram|Alpha would be an appreciated feature to have. Some participants indicated an existing high comfort level using tablets with a stylus. "I'm a paper/pen person and like using a tablet. I use a Surface Pro so it would be nice to draw the math instead of type the math" (P8). Many participants agreed that pen recognition would be an ideal way of interacting with CAS or online software applications. When asked about concerns which arise when relying on pen recognition as a form of input, most participants were quick to point out the accuracy of the recognition. "My biggest concern is the ability to recognize input. My writing isn't always the neatest. It would be frustrating if it couldn't understand it on the first couple of tries. If that were the case I'd quit and go back to typing" (P14). Therefore, ensuring a consistently high level of accuracy when recognizing a user's pen input appears to be crucial in order to increase or even maintain a larger amount of active users for an application reliant on pen recognition.

2.4.5 Reliance on pencil/paper

With a majority of the discussions having been centralized around CAS use, it was important to be able to determine the reliance that still exists when using pencil/paper. Specifically, what concerns arise when transitioning from math being done on paper to being done with CAS? This aspect of using pencil/paper demonstrates that some students learn best by trying problems on their own, making mistakes, and then correcting their mistakes. While software solutions may exist to simulate a pencil/paper feel, it would

be difficult to make up for the comfort level that would be lost by some students if pencil/paper were no longer an option. It does not appear that the participants are interested in moving away from pencil/paper entirely, although there still exist some clear advantages for transitioning to an increased usage of CAS. Regardless of the quantity or quality of software systems available to students, most participants were apt to do math "by hand" for the rough work stage of completing math assignments. In some cases, this even extended to completing assignment problems with groups of friends. "We would use pencil/paper and a whiteboard when first going through the problems" (P8). Reasons for doing math by hand for rough work, as opposed to using software, were often related to an organization of thoughts. "It's good to record steps and your thinking process to look back later" (P9). One participant pointed out that recognizing pen input for jotting down rough work can be an unnecessary concern. "Paper is my free canvas. If software is actively trying to recognize my rough work, it can be an added stress" (P15). Discussions with participants on the active or continued role that pencil/paper will play when completing math assignments shows a strong indication that pencil/paper will not be entirely replaced by software anytime soon.

2.4.6 Complexity of input

The complexity of the input had a great impact on whether the participants used a certain tool or not. Pencil/paper appeared to be most ideal in this situation since participants claimed that doing the steps on their own was crucial to reinforcing a given topic. This is likely what is required of the students anyway, since the majority of their assignment questions are computation related. Participants who had used Maple for assignments claimed that Maple inputs were difficult in general since you still had to learn the syntax. While the complexity of expressions would not affect the overall syntax, it became hard to visualize exactly how Maple was interpreting their query. A similar issue was also expressed with Wolfram|Alpha. Despite not having to learn any strict syntax rules, Wolfram|Alpha queries are input as one single line of text. A user will not see how Wolfram|Alpha interpreted their query until the output is produced. This became a larger issue as the complexity of inputs increased, often caused by queries containing large fractions and exponents. "If the input contained complex or nested fractions with lots of brackets, it can be daunting to write into Wolfram" (P14). Discussions regarding using CAS with complex inputs related directly back to pen recognition as a form of input. "For hard questions, I would prefer drawing my input" (P13). It seems that pen recognition is most useful when the input is considered difficult to type, due to either ambiguity or the complex nature of properly balancing brackets. On the other hand, most participants made the argument that they

wouldn't want to use pen recognition for very basic input. "If I was just trying to integrate e^x, it would be easier to just type it rather than draw it" (P15). These discussion were beneficial in isolating the conditions which pen recognition would be an appreciated or useful feature for interacting with CAS.

2.4.7 Cost and accessibility

The cost and accessibility of CAS had a minor role in how participants viewed the different tools for completing math assignments. Pencil/paper clearly did not face any limitations in this category since it is extremely cheap to use and is not location dependent. Maple was given negative feedback, mostly in terms of accessibility. For the participants who completed assignments which required using Maple, their university had purchased a site license for students to use. Students therefore did not directly have to pay to use Maple, however they had to be logged onto a school computer on campus to use it. This became an annoyance for students who would rather complete their assignments from home. Wolfram|Alpha was more appreciated in this area since it is a web/mobile application that can be accessed and used for free from any location. No major concerns were discussed about losing internet connectivity.

2.4.8 Software suggestions and openness to change

Based on the discussions regarding pencil/paper and CAS use, participants were asked what suggestions they had to further improve math software in order to eliminate some of the limitations that were discussed. A lot of these suggestions were derived from the already mentioned themes and topics of discussion. Pen input was the most consistent suggestion. Since Wolfram|Alpha was used by nearly all participants, and all participants also agreed that input was its biggest limitation, adding pen input to Wolfram|Alpha was a very popular suggestion for improvement. Some participants also brought up that being able to convert pen input into various syntax forms (Maple, Mathematica, LaTeX) would be very helpful when interacting with other CAS. Ensuring that the software had a minimum set of features in order to improve the ease of use was another common improvement that was suggested.

After discussing improvements that could be made to math software and CAS, all participants were then asked if they would change how they complete assignments in order to include this software into their routine. All of the participants stated that they would

be very open to the idea of using improved applications for completing assignments, given that the software includes some of the improvements that were suggested.

Chapter 3

Web implementation

The main contributions of this thesis surround the web implementation of the MathBrush application. MathBrush has been previously developed for iOS and Windows devices, specifically. When making the transition from these platforms to the web platform, there were certain considerations that had to be made. One of the most prominent of these was examining how the architectures of the platforms differed. Another large consideration revolved around code re-usability. How much code from the iOS and Windows MathBrush codebases could be used for the web implementation? This leads to a final major consideration regarding the compatibility of certain components, such as libraries, server access, etc. Can we assume that libraries which work on iOS will also work on the web?

This chapter aims to examine all major aspects related to the development of the MathBrush web application. These include: architecture, limitations, development environment, code compatibility, and remaining challenges.

3.1 Application architecture

Designing and implementing an efficient architecture for an application is a very crucial step in development. Some of the considerations to keep in mind when designing an application's architecture are: simplicity, compatibility, using modern technologies, latency, and maintainability. Since the existing MathBrush iOS and Windows platforms make use of an already existing architecture, it would be ideal for the MathBrush web platform to use a very similar architecture. However, several architectural challenges exist when attempting to migrate an application between platforms. These challenges are examined

in order to support any modifications that are made in order to design a functional web architecture for MathBrush.

3.1.1 Mobile architecture

Before designing the architecture of the MathBrush web application, it was valuable to first understand the architecture of the existing MathBrush platforms. In particular, the iOS platform was examined. Figure 3.1 details a high-level diagram outlining the main architecture supporting the MathBrush iOS application.

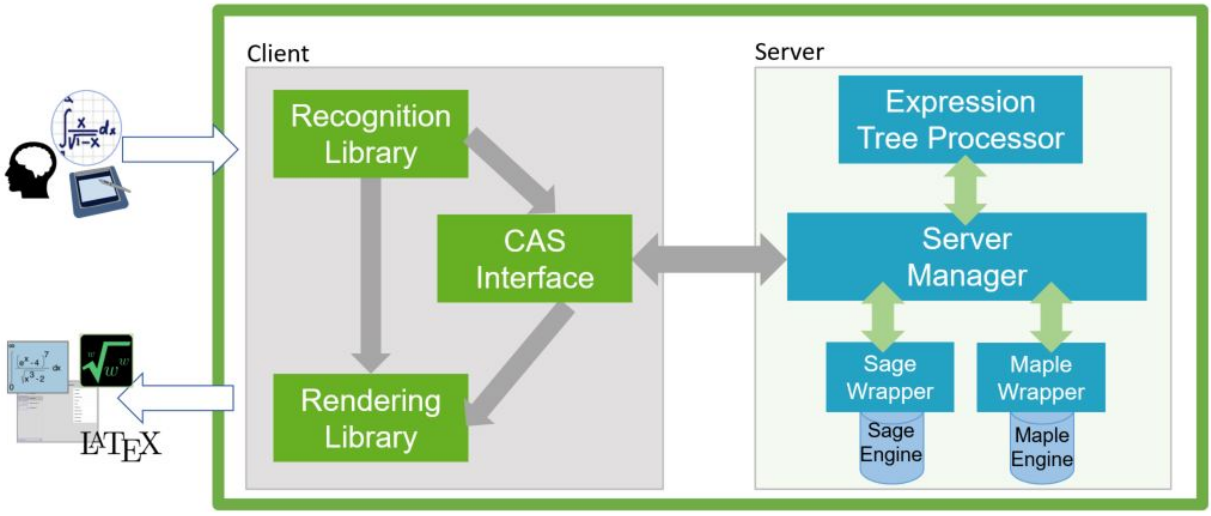

Figure 3.1: MathBrush iOS architecture

This diagram is useful in illustrating the relationship between the user's input, the components on the client and server, as well as the output returned back to the user. The main input from the user is in the form of drawing ink (i.e. math expressions). Immediately after acquiring this input, the application is responsible for recognizing the ink using a recognition library. This recognition library is written in C++, and is examined in more detail later. The output of the recognition library is an expression tree, representing the recognized expression. In order to display this result in "pretty print" to the user, the expression tree is passed to a rendering library. This rendering library then displays the result to the user, and allows the user to modify or correct this result if necessary. Both the recognition and rendering libraries occur on the client side of the application. In terms of linking CAS (such as Maple or Sage) to this system, that is when the server-side

functionality is used. After an expression tree is returned from the recognition library, the user has the option of passing that expression tree to a CAS interface. This CAS interface provides the user with a list of possible operations (such as evaluate, differentiate, simplify, etc.) to be performed on the recognized expression. The CAS interface communicates via a socket connection to a server manager on a remote server. The server manager is then able to communicate with a Sage or Maple engine via its respectful wrappers by passing the expression tree object and the operation to apply to it. The result is returned back to the server manager, and then converted to a format that is readable by the CAS interface. From there the result is returned back to the CAS interface on the client, to be rendered and displayed to the user.

3.1.2 Web architecture

As mentioned above, the goal when designing and implementing the architecture for the MathBrush web application is to keep it as similar as possible to the iOS and Windows architectures. With that being said, major limitations in migrating between platforms can warrant architectural changes to be made.

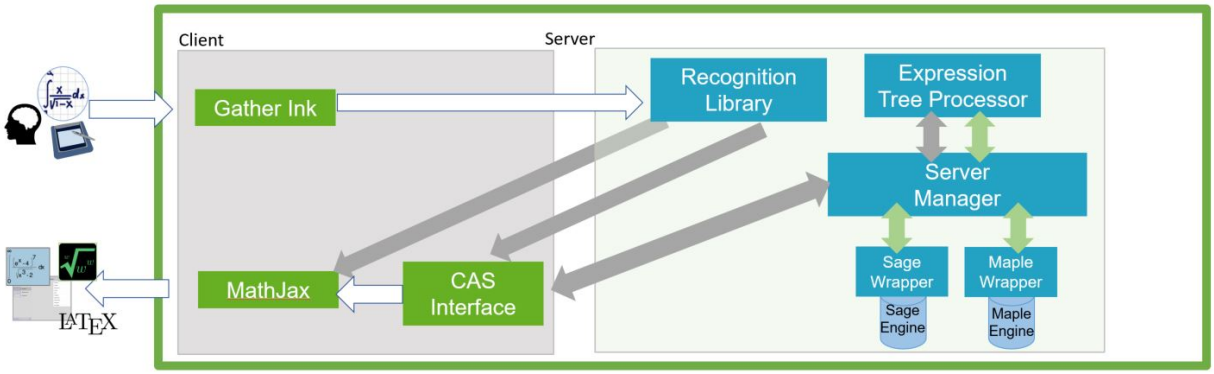

Figure 3.2: MathBrush web architecture

Figure 3.2 illustrates the architecture supporting the MathBrush web application (at the time of development concluding for this thesis). The noticeable differences between this and the iPad architecture are discussed and explained during this chapter. In summary, the main differences when moving from the iPad architecture to the web architecture for MathBrush are:

1. The gathering and management of ink is manually added to the client

2. The recognition library is moved from the client to the server.

3. The rendering library is removed and replaced with the MathJax JavaScript library.

3.2 Web vs iOS MathBrush features

The web version of MathBrush aims to be as similar to the iOS version of MathBrush as possible, in order to main a high level of consistency among platforms. However due to technical challenges and time constraints, the MathBrush web version only offers a subset of the MathBrush iOS features. Future developments towards the web version of MathBrush will seek to add more features already available in the iOS version.

Every feature available in the MathBrush web application is also available in the iOS MathBrush application, with the exception of the Wolfram|Alpha compatibility (discussed later). Some of the remaining features which are available in the iOS version of MathBrush, but not available in the web application are:

- Rendering expressions directly
- Saving/sharing expressions
- Managing user accounts
- Help menu/demonstration
- Graphing support
- Recognition training

3.3 Existing MathBrush libraries

Managing an application distributed across multiple platforms is not an easy task, mainly because this is often composed of managing multiple codebases. This means that making a change to the codebase for one platform usually requires making an equivalent change in the codebase of the other supporting platforms. It then becomes a large priority to maintain a certain level of consistency among these platforms during development and maintenance. To alleviate some of these struggles, the use of pre-compiled libraries, or shared resources,

can become an effective means of sharing code among different codebases. As a result, a modification made to the library will impact all codebases the same way.

The MathBrush application has made use of libraries during development of the iOS and Windows versions. The most prominent libraries used are the recognition library and the rendering library. For the development of a web version of MathBrush, it only makes sense for the application to also make use of these resources. Thus all three versions of MathBrush share the same libraries, and maintain a high level of code consistency even when changes are made to the libraries.

Attempting to make the recognition and rendering libraries compatible with a web version of MathBrush presented a large amount of challenges. A high-level reason for this was due to the programming languages that support each of these MathBrush platforms.

3.3.1 Platform programming languages

Each platform that MathBrush is available on uses a different set of languages, specific to the architecture of the devices which the application will run on. The compiled libraries have all been written in C++, however the front-end languages that support each platform are different.

iOS

Objective-C has been the main language used in developing applications for OSX and iOS platforms. Recently, a newer programming language called Swift has become the more supported language for these development environments. The development of MathBrush for iOS began before Swift was introduced, and so Objective-C remains the language of choice for the iOS version of MathBrush.

The front-end of the MathBrush iOS application is thus written in Objective-C. The communication between the Objective-C front-end and the compiled C++ libraries is fairly straightforward. Wrapper classes allow for functions in the libraries to be easily called from the front-end, as well as the passing and returning of objects. Thus there does not appear to be any architectural challenges when calling these libraries from the Objective-C front-end.

Windows

Development of Windows applications is supported by its own .NET framework. This framework makes use of the programming language C# for its main front-end development.

The communication between the C# front-end of MathBrush with the compiled C++ libraries is also fairly smooth. Functions from these libraries are easily callable from C#, and as a result provides a fairly straightforward incorporation of the C++ libraries with the Windows version of MathBrush.

Web

The architecture and framework surrounding web technologies present several differences compared to native development environments (i.e. iOS, Windows, etc.). This is mainly due to native applications being compiled specifically for the architecture of the devices which they run on. Web applications, however, have the potential to run on any device with a web browser.

The front-end of basic web applications are in the form of HTML/CSS/JavaScript, representing the outline, style, and functionality, respectively. In terms of the user interface, there are no real limitations in developing a MathBrush web version. The iPad and Windows versions each have their own user interface developed in Objective-C and C#, while the web version's interface can make use of HTML/CSS/JavaScript to do the equivalent.

The main difficulties with the web version are mostly related to functionality, specifically the ability to call the compiled C++ libraries. The frameworks surrounding the iOS and Windows applications made this task fairly trivial for their respective development environments. Calling compiled libraries from JavaScript is by no means impossible, but it can present some challenges, mainly depending on the language which the library code was compiled.

The Django web framework and the Apache Struts web framework are common tools used to easily create a smooth line of communication between their respective languages (Python and Java) and the front-end of the web application. In the case of developing a web version of MathBrush, it is crucial to create an architecture which supports an efficient level of communication between the front-end of the web application and the compiled C++ libraries it will rely on.

Surprisingly, there was very little documentation on modern tools which support calling functions compiled in a C++ library from JavaScript. While many companies rely on legacy code which supports this architecture, it does not appear to have much common usage today. This can be challenging when attempting to build a modern web application which must be able to call a C++ library. Scott Frees, a freelance developer and college professor, recognized the lack of documentation on this topic and published a book and a series of blog posts which provide relevant and useful information on integrating C++

libraries with modern web applications [19]. This provided a fantastic place to start as it laid out the web technologies required to support this integration. Frees was even kind enough to quickly reply to e-mails concerning any questions I had on that topic.

While other web frameworks and languages used for the web version of MathBrush are discussed during this chapter, it is important to note that the communication between the front-end of the application and the compiled C++ libraries are done through JavaScript.

3.3.2 Recognition library

The recognition library is a compiled C++ library used for all platform versions of MathBrush (static library for iOS and dynamic link library for Windows). This thesis does not intend to modify the recognition library (a.k.a. the recognizer), but simply to treat it as a black-box which the web application can call.

The recognizer is executed on the client for both the iOS and Windows platforms of MathBrush. Therefore the application does not require any type of network connection in order to use the recognizer. While the C++ code for the recognizer is the same for each platform, they are compiled separately to support the architecture of each platform. As mentioned, the major challenge with using the recognizer for the web version of MathBrush is finding an efficient and modern way of calling the the C++ library from JavaScript. There exist three different possibilities in particular for accomplishing this [19]. All three of these options require the web application to be supported by Node.js. Node.js is a JavaScript environment used for the purpose of integrating server-side code. This means that the recognizer would need to be located on an external server which would be called from the client (JavaScript) using Node.js. Calling the recognizer on a remote server can draw some immediate concerns. The first concern is that a network connection is now required to call the recognizer, which is not the case for the iOS and Windows platforms. Since this is for a web application, any user who can open the MathBrush web application must already have an internet connection. Therefore this added constraint of the recognizer requiring a network connection for the web version of MathBrush is not a large issue. A second possible concern is latency, such that returning results from the server can be much slower than the recognizer being run directly on the client. This is a valid concern, as there is no guarantee on the strength of any given user's web connection when using the application. One argument could be made that both the speed and coverage of wireless networks will continue to drastically improve, and thus the chance of a user having a weak internet connection will continue to be a fading issue. On the other hand, there may actually be more consistency in the recognizer's efficiency when it is run on a remote

server. When the iOS MathBrush application is considered, it is known that the only piece of hardware which that application will be run on is an iPad. Therefore, the hardware specifications for iOS clients is not a concern since it is known that the recognizer can be executed efficiently on the iPad. However, there is an extremely large range of devices on which a web version of MathBrush can be run. As a result there is no guarantee on the hardware specifications for the device a client is using to access MathBrush. While this is also a fading issue since the hardware on mobile devices continues to improve over time, it is important to note that the computing power of each web client can still differ drastically. By placing the recognizer on a remote server, the computation time for the recognizer to generate some output will not be dependent on the device each client is using. One discrepancy will be the client's upload/download rate on their network, which could nearly be considered negligible given the relatively little amount of data being passed to the server and back. A more relevant concern could be how well the application scales as a growing number of clients are handled by the server. While this thesis does not further investigate the scaling of client management, it is an important consideration for future hosting and management of the application.

Given that calling the recognizer on the server should not cause any performance issues, one of the three options on integrating the C++ library with Node.js needs to be decided. The three general ways for implementing this integration are [19]:

1. Automation: Call your C++ as a standalone app in a child process.
2. Shared library: Pack your C++ routines in a shared library (DLL) and call those routines from Node.js directly.
3. Node.js addon: Compile your C++ code as a native Node.js module/addon.

The main factor considered when choosing between these options was whether there is access to the source code or just the binary. Since this project does have access to the code of the recognizer, it was beneficial to make use of this. Option 1 (automation) does not require access to the raw source code, but relies on the C++ code to be run as a standalone app. The recognizer is designed as a library, not a standalone app, so this would not be a good option. Option 2 (shared library) involves compiling the C++ source code as a dynamic/shared library. A foreign function interface module can be used to call functions in the library from Node.js directly. This requires access to the source code so that the library can be properly linked and compiled to support the platform for which it will be hosted. Option 3 (Node.js addon) also requires access to the source code (which isn't an issue) and allows the creation of a Node.js addon using the Google V8 APIs. Both Options

2 and 3 have the potential to work properly. The decision was made to pursue Option 2 (shared library) since the iOS and Windows platforms already make use of shared libraries.

Foreign function interface

Scott Frees details how to call a C++ DLL from Node.js using a basic example to describe this process [19]. In the example application, a user is prompted to enter an integer to the JavaScript front-end. The application then displays all of the prime numbers less than or equal to the integer that the user had input. The calculations for constructing this list of prime numbers are done in a function within a C++ shared library. In order to call this function from the shared library, Node.js makes use of a Foreign Function Interface (FFI). FFI is an addon to Node.js used purely for calling and loading shared libraries through JavaScript. In order to use the FFI module, the names of the functions being called, as well as the types of input/output for each of those functions must be declared. Data obtained from the client can then be passed to one of these functions in the shared library. The results are returned to a JavaScript variable, taking the form of the type of object the function produces. The front-end JavaScript can then access this returned value, and use it however fit (e.g. displaying it).

In terms of using the recognition library, the Node.js client passes a list of points that the user drew on the front-end of the application. This list of points is passed via the FFI to the main recognition function in the C++ shared library. The recognizer produces an ExpressionTree object which represents the newly recognized expression. Another function is then used to convert the ExpressionTree object into a String representing the latex form of the expression. This String is what gets returned back to Node.js, and is displayed to the user (via MathJax). This same process is used for deleting strokes, clearing ink, and generating unique user keys (discussed more in the client management section).

3.3.3 Rendering library

The rendering library is also a C++ library used for the iOS and Windows versions of MathBrush. The intent of the rendering library is to both display mathematics (from the recognizer or CAS output) as well as allow the user to modify these expressions directly. The rendering library appears to the user as more of a front-end feature than a back-end feature. The front-ends of the iOS and Windows applications rely on different languages for controlling the front-end (Objective-C and C#). In order for the rendering library to work on both of these platforms, a graphics context is passed between the front-end

of the application and this rendering library. Thus the rendering library can display or manipulate graphics on the iOS and Windows platforms via a pointer to the front-end of their respective applications. Objective-C and C# support this low-level architecture for using shared libraries to control the graphical user interface (GUI).

The challenge arose when attempting to use this functionality on the web. Frees briefly discusses incorporating C++ DLLs into the GUI of a web application:

"If you are looking at a program written with only a graphical user interface... well then you are in a world of pain. It's likely you are going to need to rewrite your application in order to make it work on the web [19]."

This warning well summarizes the challenges faced when attempting to use the rendering library with the MathBrush web application. The problem basically comes down to the form of data which can be passed between the JavaScript client and the server-side rendering library. Recall that the Foreign Function Interface is used to pass data between Node.js on the client and the shared C++ libraries on the server. This works well with the recognition library since trivial types of data are being exchanged (such as a list of integers representing the ink, a String representing the LaTeX expression, etc). For the rendering library to work as intended, a graphics pointer would need to be passed from the shared library to the JavaScript front-end. This would theoretically allow anything displayed in the rendering library to also be displayed on the front-end of the web application. Unfortunately, this type of connectivity using graphics context pointers is not supported by the Node.js/FFI architecture. Without a working pointer to the display canvas of the application, the rendering library for the most part becomes unusable.

Given this large challenge of connecting the rendering library to the MathBrush web application, a few possible workarounds were considered:

1. Switching to an ASP.NET framework
2. Creating a front-end rendering library "interpreter"
3. Displaying results using MathJax

ASP.NET framework

Once it was made clear the rendering library would not be compatible with the current Node.js architecture of the MathBrush web application, use of the ASP.NET framework

was considered. The .NET framework is the Windows framework used for developing desktop and mobile Windows applications, such as MathBrush. The ASP.NET framework is an extension of the .NET framework used specifically for developing web applications. Changing to the Microsoft environment of the ASP.NET framework would involve redoing several parts of the application, including: calling the recognizer, collecting ink, and compiling the DLLs. The first goal was to gather ink input from the user using an ASP.NET web application. Windows has some built-in ink libraries as part of the .NET framework, and the Windows version of MathBrush makes use of this. Ideally the web application should be able to make use of these same libraries using the ASP.NET framework. Surprisingly, this was a much more difficult task than anticipated. The same process for adding the ability to draw ink on a .NET application could not be repeated for an ASP.NET application. The most recent Microsoft documentation on how to accomplish this was published in 2005, 12 years before the writing of this thesis [11]. Much of the documentation and technologies discussed were outdated, and Visual Studio (Microsoft IDE) had gone through several changes. While concurrently trying to add ink management to the ASP.NET application, time was also being spent on connecting the rendering library. Many issues related to the Visual Studio development environment along with some incompatibility issues with the rendering library and the ASP.NET framework slowed down this process. The decision was made after spending several weeks with the ASP.NET framework to return back to the original Node.js framework. While I am sure implementing MathBrush as an ASP.NET framework is by no means impossible, the large amount of challenges in doing so along with limited time constraints did not seem like a viable option to pursue for now.

Rendering communication

By choosing to stick with the Node.js framework for the MathBrush web application, it was understood that the rendering library would not be able to work as intended with the application. However, a minimum requirement for the application is to display the recognized and computed expressions returned from the recognizer and CAS. Additionally, it would be ideal for the user to have the ability of modifying the recognized expression in case it was improperly recognized. Since these features were all covered by the rendering library, other alternatives needed to be considered. One idea is to still use the rendering library purely for computations, and render the results of those computations on the front-end of the application.

The rendering library is structured such that each symbol (operator, variable, or operand) is surrounded with a virtual box. This virtual box has the minimal dimensions required to fully enclose each symbol. Sub-expressions (containing grouped terms)

are also surrounded by virtual boxes with minimal dimensions. The rendering library displays results (e.g. recognized expression) by grouping all of these virtual boxes in such a way that the expression appears in "pretty print." Modifying the expressions is done by having the user select one of these virtual boxes (either a symbol or sub-expression), and the user will then be given the option of selecting a different recognition for the virtual box.

For the Windows version of MathBrush, the pointer to the front-end of the .NET application allows the rendering library to draw the recognized expression (structured as a set of virtual boxes) directly to the front-end of the application. Since this type of connection via a pointer is not supported by the Node.js architecture, an idea was for the rendering library to simply return data pertaining to the sets of virtual boxes. For a given recognized expression, the rendering library could simply return the dimensions, locations, and contents of each virtual box composing the recognized expression. The front-end of the application (JavaScript) could then interpret this information and display it accordingly. While this may be a theoretically good solution, there are some realistic challenges to consider when attempting to implement this. One major challenge is simply the complexity and amount of corner cases to consider when implementing an interpreter for the rendering library. Cases such as line breaking, large output, handling the recursive behaviour of the library (e.g. multi-layered fractions) can all be quite difficult on their own to develop. Another challenge would be how to modify the recognized expressions. Since the pointer connectivity to the recognized expression is lost, modifying the expression directly becomes more difficult. One possibility would be to assign a unique identifier (UID) to each of the virtual boxes. The UID for the virtual boxes would be consistent between the rendering library and the front-end of the application. When a user selects one of virtual boxes, the UID of that box is passed back to the rendering library. If a change is made to the recognition, the ExpressionTree object would need to be updated based on which virtual boxes have changed. There would likely still exist some difficulties in modifying the ExpressionTree based on using UIDs from the recognized expression, since these types of manipulations were not intended.

A similar idea is to generate a series of PNG images on the server, where each image would represent an individual virtual box. These images would then be assembled on the client, once again relying on the size and positioning of each image in order to form correctly. UIDs would once again be used in order to maintain a connection between each image (representing a virtual box) and its corresponding value in the ExpressionTree object. The decision was made that this would be too complex of a solution to implement in a limited time frame, although this idea would still have potential as a future consideration.

MathJax

A final idea to act as a replacement to the rendering library was simply to display the recognized and computed expressions using MathJax [6]. MathJax is a popular JavaScript library used for displaying mathematics on web pages. By providing MathJax with either the MathML or LaTeX of a math expression, the MathJax engine will then be able to display the "pretty print" of that expression directly onto an HTML page. The MathJax documentation was very helpful in getting started, and thus this option was chosen as the means of displaying mathematics in the MathBrush web application.

By relying on MathJax for displaying the recognized and computed expressions for the application, a few concerns were examined:

1. Reliance on third-party software: The recognition and rendering libraries used for MathBrush were developed for the sole purpose of supporting MathBrush. Therefore changes to these libraries can easily be made to remain consistent with the needs of the MathBrush application. By using Mathjax, a third-party piece of software, that control is no longer present. MathJax cannot be altered to meet the specific needs of MathBrush. This can be a concern since there is now a certain dependence that MathBrush would have on the viability of MathJax. Given its popular usage and large amount of users, there are no imminent concerns regarding MathJax being shutdown. With that being said, if one of the goals of MathBrush is to use as much internal code as possible, then MathJax may only be a short-term fix for displaying expressions on the web application.

2. Displaying on canvas: The output of MathJax is mainly intended to be displayed on basic HTML pages. This is often done by embedding the LaTeX or MathML in the body of the HTML page, and referencing the MathJax engine in the JavaScript. The front-end of the MathBrush web application makes great use out of the canvas object (discussed in more detail under the Canvas section). Essentially, the canvas is meant to be drawn on (e.g. shapes, lines, pictures). Drawing on the canvas is done from JavaScript, where attributes of what is being drawn are provided. MathJax does not have the ability to be "drawn" on the canvas. However since the entire front-end of the application is a large canvas, it is a necessity for MathJax to still be displayed on the canvas. The solution is to overlay the MathJax output *on top* of the canvas. This gives the appearance of MathJax being drawn on the canvas, while remaining as a separate HTML element.

3. Line breaking and scrolling: Line breaking is a common challenge for the development of any computer algebra system. By using MathJax to display the mathematics, any

types of alteration to the output is limited by supporting features of MathJax. Luckily, MathJax has a built-in line breaking function. For example, if a polynomial was being displayed in the result pod of the application, MathJax's line breaking would know where to add the line breaks between the terms in order to avoid horizontal overflow. With that being said, there are also some spacing limitations that occur from using MathJax. If a matrix is being displayed that exceeds the size of the result pod, the line breaking feature of MathJax will not affect the output. Getting MathJax to even shrink its output to the maximum size that will fit into the result pod is also a seemingly difficult task. These problems relate back to the limitation surrounding the reliance on third-party software. Active work is still being made on better controlling the size and spacing of the MathJax output.

3.3.4 Connecting to CAS

The ability to allow users to connect to computer algebra systems is a crucial part of MathBrush. While the recognizer is designed to recognize input from the users, the recognized expressions only become useful once operations can be applied to them. A typical user of MathBrush would draw some kind of input (e.g. a matrix), and then choose from a list of operations to apply to that input (e.g. inverse, determinant). Traditionally when using Maple or Sage directly, the user would be expected to provide the syntax required to compute some sort of typed input. By connecting MathBrush to these CAS, a user can instead achieve the same results without having to provide any syntax.

The three computer algebra systems which can communicate with the MathBrush web application are:

1. Maple
2. Sage
3. Wolfram|Alpha (Mathematica)

For the iOS and Windows versions of MathBrush, all CAS operations are performed on a remote server. The web version also relies on this server to perform CAS operations. Due to the lack of a licensing agreement with Maple, the web version of MathBrush does not include an option to compute results using Maple (although the development version offers this choice). Instead, Sage is the default CAS used to perform various computations on the recognized expressions.

The connection between the MathBrush web application and Sage is fairly straight forward. The recognized expression and the operation the user wishes to perform (e.g. evaluate, factor, simplify, etc.) are passed to a function on the server (in the same way the recognizer is called). From there, the server-side function then passes this information to the CAS server where a result is computed. The result is then returned back to the server-side function, which then returns the LaTeX representation of the result back to the front-end of the application. The CAS server also contains a Sage wrapper in order to handle form conversions between the recognizer and Sage.

The other computer algebra system which can communicate with the MathBrush web application is Mathematica (via Wolfram|Alpha). The formative study for this thesis indicated that many students make use of the Wolfram|Alpha website when completing math assignments. Since input into Wolfram|Alpha is mainly done using a single text bar, it would be ideal to include pen recognition as a form of input. One initial idea was to do this in the form of a browser extension. Chrome extensions and Firefox plugins are small applications that are loaded as part of the web browser. An ideal scenario would be where a student uses one of these browsers to visit wolframalpha.com. Then instead of typing their input in, the student could simply hit the MathBrush icon in their browser, and then be prompted with a canvas in which they can draw their input. The recognized expression from this input would then populate the search bar of Wolfram|Alpha, and thus allow the user to make a query using the MathBrush recognizer. Unfortunately, time constraints prohibited this from being implemented. Instead, a button was added to the user interface of the MathBrush web application which still allows some connection to Wolfram|Alpha. After a user has drawn their input in the web application, the user can simply select the Wolfram|Alpha icon in the recognized expression pod. This selection will open Wolfram|Alpha in a new browser tab, and will pass the LaTeX of the recognized expression to Wolfram|Alpha. The natural language processing (NLP) of Wolfram|Alpha is designed to handle some forms of LaTeX as valid input. This same process can also be repeated for the result pod of the web application, in which the LaTeX of the computed result can also be passed to Wolfram|Alpha. Depending on the LaTeX being passed to Wolfram|Alpha, there may often need to be some String manipulations applied to the LaTeX before being sent. For example, Wolfram|Alpha does not recognize the "bmatrix" notation for matrices, so some pre-processing is done on the LaTeX string to convert the LaTeX into a format that is understood by Wolfram|Alpha.

3.4 Ink management

One of the main challenges faced when developing the MathBrush web application surrounded managing the ink input. There were no external JavaScript libraries used or available which would be able to handle the needs for the different types of use cases required for MathBrush. As a result, the management of the ink was created from scratch using JavaScript. The types of operations to consider for managing the ink input are:

1. Drawing strokes
2. Extracting all points from each stroke
3. Delete stroke(s)
4. Undo stroke
5. Undo delete
6. Clear ink

An HTML5 canvas is the main component used for drawing ink input into the web application. A canvas is often used for drawing lines, shapes, and graphics into a predefined space in an HTML page. Action listeners within JavaScript can detect the location of the cursor on the screen, as well as whether the mouse is clicked down or up. These action listeners provide support for drawing ink directly on the canvas. Theoretically, the application would be able to track every coordinate that the cursor has visited with the mouse clicked down, and draw a single point at each of these coordinates. This would result in displaying the ink where the user has just drawn their input. The problem is that a user can draw ink faster than the JavaScript action listener is able to detect every coordinate that the cursor has visited. As a result, only a series of non-connected points would be drawn to the canvas. This is problematic for two reasons:

1. The ink displayed does not look smooth. There would be several gaps between the points drawn which make up the ink.
2. The recognizer accepts a list of points as its input. With several points missing from the input, the recognizer is not likely to give correct results.

The solution to this problem revolves around the idea of drawing a series of small lines, instead of points. Essentially, a connect-the-dots pattern is used with the series of non-connected points in order to fill in the gaps between the points. When drawing the ink on the canvas, a series of lines are thus drawn instead of a series of points. This gives the visual appearance of "complete" ink, and does not contain any unintentional gaps. As for using the recognizer, the points which fall on the lines being drawn need to be calculated. The Bresenham line-drawing algorithm is used for this exact purpose [18]. The coordinates of these calculated points can then be passed to the recognizer for a more accurate recognition.

Every stroke that is drawn on the canvas is therefore stored in two formats. One format is a list of the small lines which make up the stroke, so that the ink can be drawn/re-drawn on the canvas as complete ink. The other format is as a list of points, so that the ink can be passed directly to the recognizer. Unique identifiers (UIDs) are assigned to each stroke, with a 1-to-1 mapping between the list of stroke lines and the list of stroke points. When wishing to undo or delete certain stroke(s), these UIDs are passed to the recognizer. The recognizer will then delete the strokes corresponding the UIDs from the expression tree, and return the LaTeX of the updated recognized expression back to the user interface. It is imperative that the stroke UIDs which make up the expression tree on the recognizer is kept in sync with the list of stroke lines and list of stroke points maintained on the JavaScript front-end.

3.5 JavaScript frameworks

The web architecture for supporting the MathBrush web application makes use of several JavaScript libraries and frameworks. A very similar architecture was used in the example given by Frees [19]. The specific libraries and frameworks used were chosen with the overall purpose of communicating with the shared C++ libraries on the server. Brief descriptions for each notable piece of web technology used in the application are described below, along with the other web technologies they communicate with. MathJax and the Foreign Function Interface (FFI) are not included since they have already been discussed.

3.5.1 Angular.js

Angular.js (often referred to as just "Angular") is a very popular JavaScript framework, used for binding data to HTML with expressions [1, 2]. Angular applications make use

of a model/view structure supported by controllers. Essentially, Angular is useful for updating information in a web application that is often prone to change. In the case of MathBrush, the recognized expression is being changed whenever the ink is modified. Angular is responsible for acquiring the changed data (from Node.js) and updating the web application to display the changes. This does not require any additional event listeners, as Angular's functionality automatically detects any changes made.

3.5.2 Node.js

Node.js (often referred to as just "Node") is a JavaScript server framework, designed to build scalable network applications [8, 9]. This allows JavaScript to be run on a remote server. For the MathBrush web application, this is especially useful for communicating with the shared C++ libraries on the server. While static HTML pages can be viewed by opening the main HTML page with a web browser, Node.js applications require the application to actually be "running" on the server. Node contains a built-in module called HTTP, which allows the transfer of data over the hyper text transfer protocol (HTTP). This module creates an HTTP server which listens to a specific server port (e.g. 8080) and gives responses back to the client. The FFI makes use of this networking to call C++ functions on the server and have results returned via Node. Node.js is also supported by another web framework called Express.js [3]. Express.js allows for easier communication between Node.js and the server, as well as for communicating the results from the server back to Angular in order to be displayed.

3.5.3 Pug

Pug (formerly known as "Jade") is not an actual JavaScript framework, but it is still important to mention as it plays a key role in the architecture of the MathBrush web application. Pug is a template engine used for writing syntax-enforced HTML. Any given HTML page can be converted into Pug, and vice-versa. The main reason for using Pug instead of HTML is that Pug communicates in a very clean way with Node.js and Express.js. Pug was also used in the discussed Scott Frees tutorial. The overall logic for constructing an HTML page remains very similar to constructing an application using Pug, with some notable syntax differences mainly related to white-space.

3.6 Client management

In order for the MathBrush web application to support multiple users, it must make sure to separate the data between each running instance of the application. Since the recognizer is a server functionality for the web application, some additional work is required to manage multiple user sessions. By default, the recognizer's shared library contains a global object which represents the expression tree of the recognized expression. When ink is added or deleted, appropriate adjustments are made to this expression tree. The LaTeX String which is often returned back to the front-end of the application is based on this global expression tree object. The problem is that the same global object is referenced from different sessions running the MathBrush web application. Consider the following use case to highlight this issue:

1. User A opens up the MathBrush web application and draws the number "4" on the left side of the screen. The recognizer updates its global expression tree object to represent "4" as the recognized expression. The LaTeX string of "4" is returned to User A.

2. User B opens up the MathBrush web application (on a different device) and draws the number "7" on the right side of the screen. The global expression tree object in the shared recognition library already contains a "4", and now takes into account the "7" that was drawn to the right of it. The global object now updates itself to represent the recognized expression "47", and returns the LaTeX String "47" back to User B. User B is now confused why "47" is the recognized expression, when all they draw on the canvas was "7".

An ideal solution to this problem would involve using sockets (or web sockets) to isolate and manage each client/server server session. This thesis did not investigate the use of web sockets to manage sessions, although it may be worthwhile to explore this option in future developments.

Instead, a somewhat cryptographic approach was taken to solve this problem, with each client (user) generating a unique key. When calls to the shared library are made, this key will also be passed to the library. Instead of storing a single global expression tree object in the shared library, a global hash map is instead used to store multiple objects. The key passed from the client will hash to a single expression tree object in the hash map. Any modifications to the ink (add/remove strokes) will be applied to this hashed object. The LaTeX being returned will also be generated from the expression tree object being hashed. The above use case could then be modified to as follows:

1. User A opens up the MathBrush web application. A call is made to the shared recognizer library which generates a unique key. The global hash map inserts a new key/value pair: the key being the key just generated, and the value being an empty expression tree object. The key is returned to the client.

2. User A draws the number "4" on the left side of the screen. The points which make up this stroke are sent to the recognizer, along with User A's unique key. User A's key is hashed to the hash map in order to access its corresponding expression tree. The recognizer updates the expression tree object to represent "4" as the recognized expression. The LaTeX string of "4" is returned to User A.

3. User B opens up the MathBrush web application. A call is made to the shared recognizer library which generates a unique key. The global hash map inserts a new key/value pair: the key being the key just generated, and the value being an empty expression tree object. The key is returned to the client.

4. User B draws the number "7" on the right side of the screen. The points which make up this stroke are sent to the recognizer, along with User B's unique key. User B's key is hashed to the hash map in order to access its corresponding expression tree. The recognizer updates the expression tree object to represent "7" as the recognized expression. The LaTeX string of "7" is returned to User B.

In terms of generating the unique key for each running session of MathBrush, a basic counter is used. Whenever is new session of the MathBrush web application is created, the counter is incremented and then used as the key. For security and privacy enhancements, improved key generation algorithms could be used to generate more complicated keys. In addition, each client's key is currently stored as a JavaScript variable on the front-end. This means that the user could inspect the elements of the HTML page and extract the contents of the key. Future work could further address the generation, exchange, and accessibility of keys between the client and server.

socket

3.7 Touch screen considerations

The iOS and Windows versions of MathBrush were designed and developed to run on specific touch-screen devices (iPad and Surface Pro). With the implementation of a web version of MathBrush, there is no guarantee on what devices the application will be run on,

or even whether or not the devices will be touch-screen. This means that some small adjustments need to be made in order to ensure the MathBrush web application is compatible with both touch-screen and non-touch-screen devices.

The most notable difference when developing for touch-screen vs non-touch-screen devices are the event listeners. For web applications that do not support touch-screen input, typical event listeners represent actions that are made with the mouse, such as: "mousemove", "mousedown", and "mouseup". These actions encompass most methods of interaction the user will have with the application. Since touch-screen devices do not often make use of gestures with a mouse, a different set of event listeners are required. The event listeners used for developing web applications for touch-screen devices are: "touchmove", "touchdown", and "touchup". In order to support both mouse and touch inputs, these gestures are treated in the same manner. I.e. "mousemove" and "touchmove" are handled as equivalent gestures.

Other smaller differences between these two types of developments are related to more low-level details in the code. Specifically, calculating the coordinates where a mouse is clicked vs where a finger touches the screen are done slightly differently. In terms of the interface, there are no differences between whether the MathBrush web application is run on a touch-screen or non-touch-screen device.

3.8 Screen density

When communicating between the user interface and the server-side recognizer, ink is passed from the front-end of the application directly to the recognizer library. This ink is in the form of strokes, where each stroke is composed of individual points (coordinates) that make up each stroke. Once the recognizer has received the strokes to be added, some additional heuristics are used to compute the recognized expression. One of the main heuristics used is the pixel density of the screen the user drew their ink on. When developing for iOS, this was a known constant value since the application had to be run on an iPad. By knowing the size (diagonal) of the iPad, along with determining the resolution of the screen, the pixels per inch (PPI) of the iPad can be easily calculated. As for devices which access the MathBrush web application, the resolution of the screen can still be determined. However, there is no way of detecting the physical size of the screen (in inches). Without knowing the exact size of the screen, it is impossible to calculate the PPI of the device running the web application. Even if it is known that the resolution of the screen is 1920x1080 pixels, it is unknown whether the diagonal length of the screen is 5 inches or 50 inches.

While it is not necessary for the recognizer to know the exact PPI of the device, the accuracy of the recognizer improves as the provided PPI value becomes more accurate. The simple solution to this problem was to provide the recognizer with an "average" PPI value. This approach has been tested on both mobile devices (with high PPI) and desktop monitors (with low PPI) with the recognizer working as intended in both cases. It does not appear the PPI value has a large impact on the accuracy of the recognizer, but is more so an optimization used for slightly improved results. Future work could include a user interface option in the settings which allows a user to manually input the physical size of their device, so that a more accurate PPI value can be calculated. A similar alternative could require the user to scale an image on the screen to approximately be a certain physical size (e.g. 10x10 cm).

3.9 Hosting

The MathBrush web application is currently hosted using Amazon Web Services (AWS). The "Elastic Beanstalk" service provided by AWS allows a Web Server to host a Node.js application. The platform of the web server is a 64bit Amazon Linux operating system, running the latest version of Node.js. The code-base of the MathBrush web application is compressed locally as a .zip file, and then uploaded via the Elastic Beanstalk console to the web server. A file called package.json (JavaScript Object Notation file) is expected to be included in the .zip file which contains the commands required to start the application. When any changes are made to the code-base locally, the same process is followed in which the code-base must be compressed and uploaded again. The Elastic Beanstalk console also provides configuration and health statistics on its dashboard in order to provide monitoring data for the hosted application.

Chapter 4

User study

The MathBrush web application was designed and developed for the purpose of using pen recognition to interact with computer algebra systems on the web. This allows the application to be available from any internet-connected device via a web browser. While this application may appeal to various types of users (such as researchers or software hobbyists), the particular type of end user this thesis mainly focused on is post-secondary math students. Many of the features and user interface considerations for the MathBrush web application were done with the intention of the application being used by students. Chapter 2 of this thesis examined the process for which math students complete their course assignments, the benefits/limitations of tools they use (such as CAS), and the role pen recognition could play when they interact with software. Discussions were often centralized on the key features used in existing software (such as Wolfram|Alpha), as well as what improvements or features would be made in a new "ideal" piece of software.

This chapter has the overall purpose of obtaining feedback regarding the user interface and overall usability of the MathBrush web application. The process for acquiring this feedback was identical to the formative study in Chapter 2, in the form of several individual semi-structured interviews. 12 of the 17 participants from the formative study were available for this user study. By using the same set of participants (with the exception of participants 1-5 who were unavailable for this user study), direct correlations can be drawn between discussions brought up by a given participant in the formative study, and their feedback after using the MathBrush web application. Conclusions of this thesis are drawn by determining the extent to which the MathBrush web application satisfies the expectations students have when using math software as part of their assignments, as well as what areas of improvement can be made to the application.

4.1 Method

While a similar process of obtaining data for this user study remained the same as the formative study, the format/agenda of this study was slightly different. Much of the discussions in this study were based on acquiring feedback from the 12 participants after having used the MathBrush web application. The same provided Microsoft Surface tablet was used by each participant. About half of the participants used a stylus when drawing on the tablet, while the other half simply used their finger. The application was hosted on Amazon Web Services, and so it was not locally on the tablet.

The typical agenda for each user study was as follows:

1. Briefly introduce the MathBrush web application and discuss its capabilities.
2. Allow the participant to freely play with the application and get comfortable with its set of features (typically lasts about 5 minutes).
3. Provide the participant with three sample computation problems to solve, covering different areas of math (calculus, linear algebra, etc.). The sample problems provided were:
   - Compute the integral:
     $$\int_3^6 \frac{x^2}{x-1} dx$$
   - Find the inverse of the matrix:
     $$\begin{bmatrix} 8 & 3 \\ 7 & 2 \end{bmatrix}$$
   - Solve the system of linear equations:
     $$\begin{aligned} 9x + y &= 6 \\ 7y - 3x &= 2 \end{aligned}$$
4. Allow the participant to further play with the application and attempt different types of input.
5. Enter discussions on the participant's feedback of the application. Most discussions centralized around highlighting benefits/limitations of the application, future improvements, and overall usability.

4.1.1 Areas of interest

Participants in this study discussed their overall opinions on the MathBrush web application, often in the form of expressing what they believed to be the strengths, weaknesses, and areas of improvement for the application. While all of this feedback is useful towards future developments of MathBrush, this chapter focuses on a subset of this feedback, as it relates back to topics discussed in Chapter 2. Specific topics of interest are:

1. Reliance on pencil/paper
2. Pen recognition
3. User interface considerations
4. Complexity of input
5. Software suggestions and openness to change

For each of these topics, feedback is acquired from the participants as it relates to the overall usability of the MathBrush web application. Benefits and limitations of the application are discussed by drawing connections to the above topics of interest. The extent to which the implementation of the MathBrush web application addresses any concerns related to these topics in Chapter 2 is evaluated.

4.2 Reliance on pencil/paper

Results from the HCI study in Chapter 2 showed that regardless of improved technology and tools, there still exists a heavy reliance on using pencil/paper when completing math assignments. Much of this was attributed to the simplicity of drawing on paper, and not having to worry about properly interacting with a piece of software. Participants in the user study were asked to describe the transition made between using pencil/paper to using the MathBrush web application. More specifically, to what extent does the MathBrush web application maintain the same level of feel and comfort that is expressed when writing math on paper? When asked this question, Participant 9 was quick to point out some interface features. "The scrap paper theme makes it feel more real" (P9), as shown in Figure 4.1. This demonstrates that the user interface of the application can play an important role in extending the "feel" of drawing on paper. If an end goal of the MathBrush web application

is to provide an easy transition from using pencil/paper to drawing within the application, these types of considerations must be made to reduce the reliance most students have for using pencil/paper exclusively.

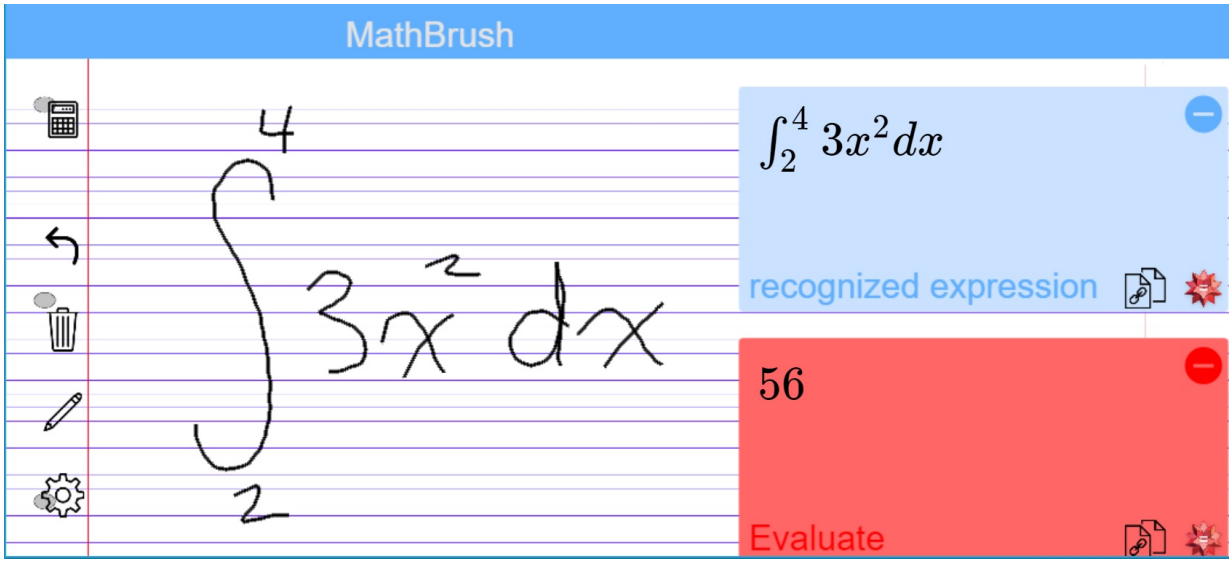

Figure 4.1: MathBrush binder interface

During development of the MathBrush web application, some discussions took place on how to display the recognized expression. The application currently displays the recognized expression in its own "pod", adjacent to the ink which the user drew. One possibility could have been to replace the drawn ink with the recognized expression in place. This diminishes the need for a separate result pod. While there may exist some benefits to this approach, one participant pointed out the reason why they prefer the current interface for displaying the recognition. "I like that the recognizer is separated from what I drew. I'm a paper/pen person, so it helps to see your own handwriting" (P8).

The overall consensus achieved from participants after using the MathBrush web application was that it is near impossible to completely replace the role that pencil/paper provide when writing mathematics. While some features of the application which attempt to bridge the gap between writing on paper and using software were appreciated, there was no indication that the application would completely replace the role of pencil/paper when completing math assignments.

4.3 Pen recognition

Participants in the Chapter 2 formative study discussed the prospect of using an application reliant on pen recognition as a form of input. While most participants agreed that pen input could theoretically be a preferred method of interacting with a math application, many also expressed concerns on the accuracy of a proposed recognizer. These concerns usually arose from participants admitting their hand-writing was not very neat, or that it would be frustrating for the application to consistently misinterpret their input.

This discussion was further continued after each participant had used the MathBrush web application to solve the 3 provided problems, as well as freely play with the application. Unfortunately, pen recognition was often discussed after participants were asked: "What are the main limitations of the application?". Nearly all participants were quick to point out what they viewed as the largest flaw in the MathBrush web application. "Handwriting recognition is the biggest weakness" (P8). The majority of the participants struggled at least a few times to have the recognizer correctly interpret their pen input. This corresponded with the accuracy concerns discussed before they even used the application. While not excusable, some of noticeable reasons for generating incorrect recognized expressions were:

- Messy handwriting: While not a surprising reason for causing an incorrect recognition, often times the penmanship of the ink input was simply quite messy. However, an important point to make is that despite the overall messiness of some inputs, in most cases the input would still be considered human readable. This demonstrates that much work on the recognizer remains in order for it to reach the recognition accuracy of a human. For some of the cases, the participants' messy handwriting was due to the high speed in which they drew their input. While increasing the speed of which they draw their input appears to negatively impact the quality of their penmanship, this should not be an issue from a user's perspective. If a participant is drawing their input very quickly when using the application, then it is likely they also write at a fast pace when drawing on paper. With an end goal of attempting to bridge the gap between drawing ink on paper and a tablet, the speed or messiness of their handwriting would ideally not impact the correctness of the recognized expressions.

- Symbols touching: Incorrect recognized expressions were sometimes due to drawn symbols accidentally intersecting with each other. When individual strokes intersect, the recognizer often interprets this as a single symbol (e.g. "x", "+", "4"). In the

case when two symbols unintentionally intersect after being drawn, this will often cause the recognizer to interpret the two symbols as one single symbol, and thus produce an incorrect recognized expression.

- Input drawn on a slant: One participant (P16) had difficulty getting their pen input to recognize correctly since it was drawn on an inclined angle. The participant likely did not consider that this would cause issues with recognition, since this may just be part of their natural writing style. Given that writing on a slant can be a common practice with paper, ideally the recognizer should be able to support this as well.

- Unexpected number of strokes to draw a symbol: The MathBrush recognizer takes into account the number of strokes that are required to draw certain symbols. This can be a very useful heuristic for improving the accuracy of the recognizer, unless a user draws symbol in an unexpected number of strokes. A common case of this was when participants were asked to draw a matrix. Nearly all participants began drawing the matrix by first drawing the left square bracket (which the recognizer assumes takes exactly 1 stroke to draw). However, at least 4 of the participants attempted to draw the left square bracket as the formation of 3 separate strokes, causing an incorrect generated recognition. A solution to this would be for the recognizer to lift the constraint on the number of anticipated strokes required to draw a symbol, although this could hinder the overall accuracy of the recognizer. A more ideal solution would involve prompting the user to "train" their handwriting within the application, so that the recognizer makes use of individual personalized heuristics to better recognize certain user's handwriting. This functionality is implemented within the iOS and Windows versions of MathBrush, but time constraints prevented it from being added to the web application.

- Spacing issues: For some types of pen input, the amount of spacing between symbols and expressions negatively impacted the calculated recognized expressions. One example of this takes place when drawing matrices. In the case of Figure 4.2, a user attempted to draw a 2x2 matrix of integers. The recognizer however interpreted it as a 2x1 matrix, combining the two columns into a single column. Had there been a greater amount of white space in between the columns, then the matrix would likely have been recognized correctly. A similar issue can also occur when drawing a system of linear equations. If there is too little white space vertically separating the individual equations, then the recognizer may attempt to interpret the equations as a single line of input.

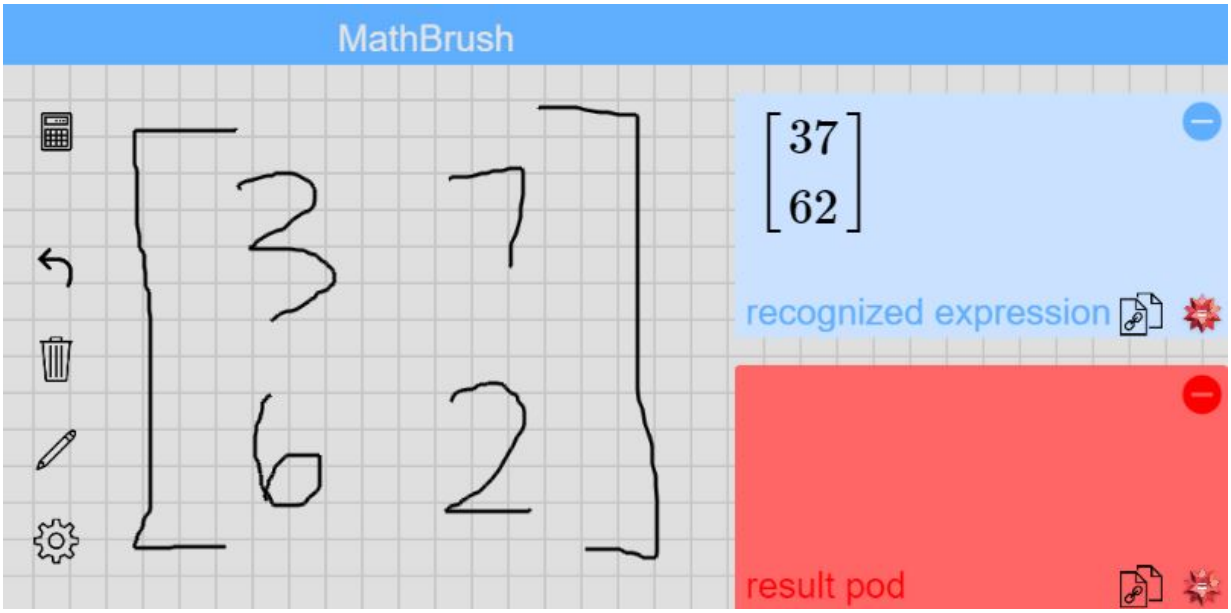

Figure 4.2: Invalid recognition due to spacing

4.4 User interface considerations

A large determining factor for whether a user will continue to be an active user of an application is the user interface. In the case of the MathBrush web application, certain considerations were made in order to design and implement a user interface suitable for post-secondary students. The Chapter 2 HCI study explored various advantages and disadvantages related to the interfaces of tools students are already using. For instance, participants who used Maple as part of their math courses admitted that the interface was quite overwhelming. This was mainly due to Maple offering a much greater amount of buttons and features than what a student will typically use as part of their courses. Wolfram|Alpha on the other hand was mostly praised for its simplistic interface, which is a single textbar to type your input. However there were some notable hindrances to this approach as well, as described in the below section.

The design and development of the MathBrush web application interface had the overall goal of being simple and intuitive to use. Unlike Maple, all features and icons within the MathBrush web application are directly intended to be useful towards novice users (students). Also given that the application is developed primarily for touch-screen devices, visually-intuitive icons and buttons are used for interaction as opposed to textual drop-

down menus. Due to the popular usage of Wolfram|Alpha by math students, the interface of the MathBrush web application grants users the ability to pass their recognized expressions or computed results to Wolfram|Alpha (in a new browser tab) for continued usage (Figures 4.3 and 4.4). Users can also select an icon to copy the LaTeX code used to generate the recognized expressions and computed results.

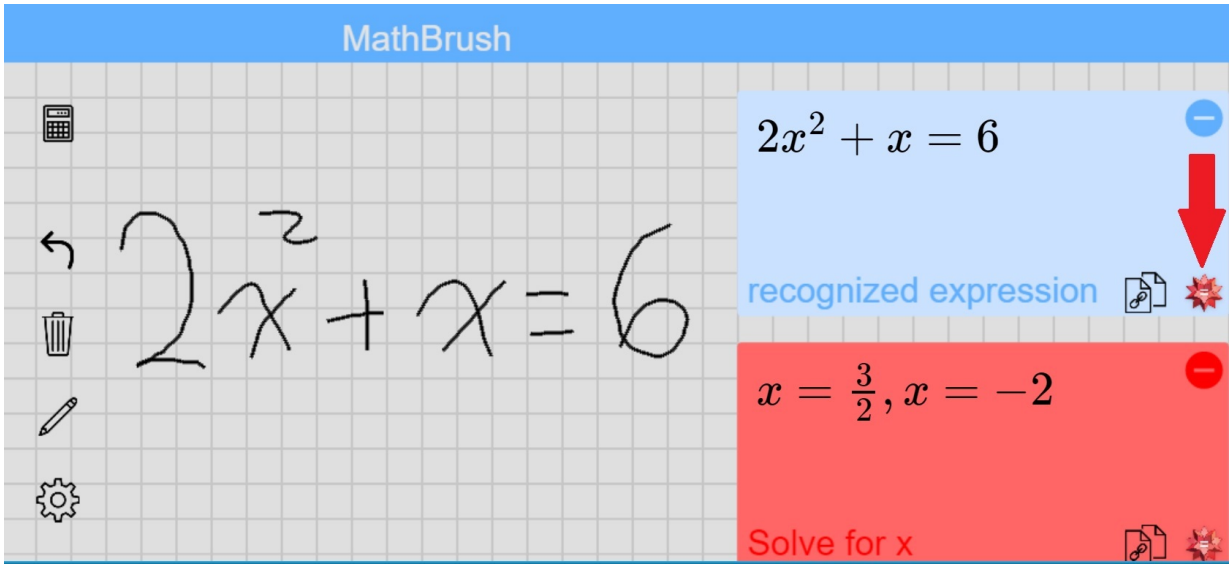

Figure 4.3: Selecting Wolfram|Alpha button

After using the MathBrush web application, participants were asked if they felt the application was intuitive to use, as well as what they liked and did not like about the interface. Nearly all participants claimed that the interface was fairly intuitive to use. The graphical icons on the left toolbar were described as "self-explanatory", and there did not appear to be much of a learning curve to properly use most of the offered features within the application. There were some exceptions, as pointed out by Participant 13: "it should have some instructions on how to use the app. Especially the scratch out feature" (P13). Adding an instructions or examples menu option to the application was a consideration at one point during development, but the decision to not include instructions was based on ensuring the overall intuitiveness of the application. If a simple, yet appealing, interface is successfully developed, then ideally this would not require a user to read any instructions on how to interact with the application. For a majority of the features of the application (such as drawing ink, changing settings, clearing ink, etc.), this appeared to be true. However, the "scratch out" feature for instance was not something any of the participants attempted to do before being showed how it works. There was no indication in the interface that the

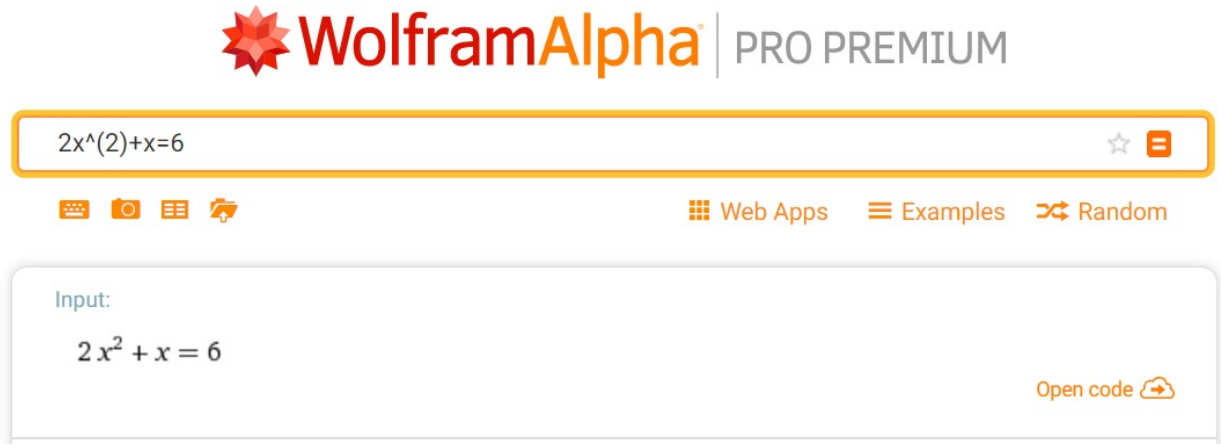

Figure 4.4: Recognized expression passed to Wolfram|Alpha

user is able to scratch out strokes or symbols that are written, and thus it is unlikely a user would make use of this feature unless they were told it exists. Therefore, in order to avoid writing user documentation or instructions within the application, it would make sense to add some sort of user interface indication describing the scratch out feature, as well as any other features users may not find intuitive in the future.

4.5 Complexity of input

One of the main factors found to have influenced the tools or resources that a student used to solve a particular problem, as discovered in the Chapter 2 HCI study, is the complexity of the problem. Some forms of input, such as the textbar in Wolfram|Alpha, were found to be useful for simple queries. Simple queries, in this case, often refers to math expressions which do not require an excessive amount of brackets to type, and are usually fairly short in length. As the complexity of the query increases, the greater the challenge it becomes for a user to accurately type in their query and have it successfully interpreted. For participants who use Wolfram|Alpha or any other online calculator or math application, most indicated that they often did have to type in their input. With the introduction of the MathBrush web application, participants and other students will now have the choice of whether to use typing or pen recognition as a form of input.

Once the participants had become familiar with interacting with the MathBrush web application, they were asked under what conditions they would use this app versus Wol-

fram|Alpha or other online tools. One participant immediately began discussing the advantages to passing the recognized expression from MathBrush to Wolfram|Alpha. "It's an easier way to input to Wolfram|Alpha" (P8). This is a reference to limitations that were discussed in the Chapter 2 HCI study, mainly the challenges associated with using a keyboard to type complex math queries. When deciding whether typing or pen input was the preferred method of interacting with an application, there was a split among the participants. Some participants admitted that they would prefer to use pen recognition whenever possible, mainly because they are already using touch-screen devices and are used to this type of interaction. Other participants argued that typing their input is preferred because they are fairly comfortable with it, and they are hesitant to rely on the consistent accuracy of pen recognition. A majority of participants, however, made the point that their preferred method of input relies entirely on the complexity of the input. If the input is fairly simple (i.e. does not require an excessive amount of brackets) to type, then most participants claimed typing would be ideal in this scenario. "I would use MathBrush for my complex inputs, and use Wolfram for simpler typed inputs" (P14). The explanation for this decision was mainly due to the simplicity and speed of typing, and that for simple queries there should not be much ambiguity when interpreting their typed input. As the complexity of the queries began to increase however, many participants claimed this is when they would prefer to use pen input to interact with an application. For complex mathematical inputs, some participants argued that it would be much faster and easier to draw the input, than it would be to figure out how to represent their input as a single line of text.

A possible interface solution was suggested by a participant, who also claimed their preferred method of input was dependent on the complexity of the problem. "An improvement could be including two options for input. One to type and one to draw" (P13). Given that the overall purpose of MathBrush is to allow pen recognition as a form of input, it seems slightly counter-intuitive to also include an option to type input (which traditional CAS have been doing since inception). However, if a main goal of the MathBrush web application is to design and implement an application that will be helpful and used by a large number of students, offering multiple methods of input should be a strong consideration. This option would allow a user to type their input into the application for simple problems, as well as draw their input as ink for more complex problems. An added benefit to this approach would be acquiring analytics on the usage of the application from active users. More specifically, determining what percent of queries made in the application were done using pen input instead of typing the input could be useful information. As well, it would be interesting to develop a model that indicates the likelihood of a user choosing each method of input given the complexity of the input. This could be accomplished by

comparing either the string-length of the LaTeX recognized expression, or the depth of the expression tree, with the mode of input chosen. A key result from this model would be determining the exact point (i.e. threshold) at which more than 50% of the users are likely to use pen recognition instead of typed input, based entirely on the complexity of the input as described above. As the accuracy of the recognizer improves over time, it would also be interesting to detect whether this threshold decreases, such that pen recognition becomes more frequently used, even for less complex inputs.

4.6 Software suggestions and openness to change

One of the last themes that were discussed with all the participants in this user study was a response to these two main questions:

1. What improvements do you suggest that the MathBrush web application should make in order to be used by more students?
2. How open would you be to including new/improved software as part of the process for how you complete course assignments?

Feedback from the first question was useful to reiterate any large concerns or improvements with the MathBrush web application that participants may have already discussed relating to previous topics of discussion. This question also gave each participant a final chance to list any other weaknesses or improvements that should be focused on in future developments of the application. The majority of responses to the first question were simply repeating some of the main ideas for improvement already discussed, such as: improving the recognizer, training the recognizer for personal handwriting (already accomplished in iOS app), rendering the recognized expression directly, adding an option for typed input, and including instructions or documentation on proper usage of the application. These ideas were suggested consistently among participants, adding validation that these suggestions should become priorities for continued development/maintenance of the MathBrush web application. As for suggestions that have not already been discussed, there was surprisingly very few mentioned. Some suggestions were related to adding features which already exist on the iOS version of MathBrush (i.e. graphing support, saving expressions), but not yet on the web application. One participant made an interesting suggestion regarding the free-form style of the pen input. "It would be nice to write in boxes for matrices and systems of equations. Integrals could still be free-form" (P13). This idea is similar to the

interface of other online web resources (besides Wolfram|Alpha) which gather user input by typing into text-boxes. For example, a user could enter a 3x3 matrix by typing atomic values into 9 separate cells. In a pen input environment, these "boxes" could more so be in the form of "suggestion boxes" as a means of encouraging users to draw their matrices in an evenly distributed way. This same principle could also be applied, as suggested, to systems of linear equations. A suggestion box could be helpful in instructing a user where to begin drawing the next equation in their system. This idea could be a plausible solution to issues identified in the "Spacing issue" section, as the suggestion boxes would help prevent a lack of or excess amount of space between symbols to cause an invalid recognition.

Responses from the second question, regarding the inclusion of new or improved software when completing course assignments, were important in determining the level of openness students had towards changing how they complete assignments. The Chapter 2 HCI study showed that students are fairly consistent in the process used for completing assignments. If a student is performing very well on their assignments, and are using a consistent process for completing those assignments, then what incentive would they have to changing that process (an homage to "if it aint broke don't fix it")? On the other hand, results from the same study revealed several limitations that exist with common tools that students are currently using to complete math assignments. Whether or not improved software will have an effect on the success of a student's assignment is unknown, however improved software can make the process for completing assignments easier for the students. When asked "would you be open to including improved software as part of your assignment completion process?", 100% of the participants all indicated that they would be very open. This is reassuring to know that students are, at the very least, open and willing to try new software if they believe it will be an improvement to what they are currently using. The MathBrush web application was purposely not mentioned in this question so that the question was focused purely on the participants' openness to trying new software, and not necessarily MathBrush. Whether or not the MathBrush web application can be considered ideal enough for students to include as part of their process for completing assignments will be evaluated in the below "usability of application" section.

4.7 Usability of application

After having discussed much of the benefits/limitations of the MathBrush web application with each participant, one major question still remained: is the application considered usable? In other words, would students find the application valuable enough to actually make use of it when completing math assignments?

More than 50% of the participants in the user study stated that they would deem the MathBrush web application (in its current state) usable. Elaborating on the definition of "usable", this group of participants indicated that given the chance they would include the application as a resource when completing course assignments. While no long-term studies have been performed to determine how well MathBrush can retain active users, the willingness of these participants to begin using the application as new users is promising. Indicating that a piece of software is usable is by no means suggesting that it is flawless, but rather that it is "good enough" to offer significant benefits to its users.

There were some participants however who did not deem the application "usable". Fortunately, the reasoning was consistent among this subset of participants: the recognizer is not accurate enough. "In its current state, I could get frustrated if it doesn't get what I write. I would try a couple times and if it keeps getting it wrong then I would stop using it" (P15). Every participant who claimed that the application is not currently usable stated it was directly due to the accuracy of the recognizer. While this thesis in particular did not focus any development towards improving the recognizer, this has been a well known priority within the MathBrush team. If future developments for MathBrush focus on specifically improving the accuracy of the recognizer, this will greatly improve the usability of the application and be a positive impact towards growing a larger user base. Since the same recognizer is used across all MathBrush platforms (iOS, Windows, web), improvements to the recognizer will benefit all running versions of MathBrush.

Quantitative data relating to the accuracy of the pen recognition when the participants completed the three sample problems is not provided. Since this thesis does not modify or improve the recognizer, this type of data is not completely relevant. However, it can be pointed out that 100% of the participants were required to correct an incorrect recognition at least once while drawing the three problems, and often times more than once.

While it is not ideal to have some participants conclude the MathBrush web application is not currently usable, it is reassuring to know that this is not due to any user interface flaws or any other limitations within the application. One of the main goals of this thesis was to design and create an intelligent web user interface to allow proper communication between the pen input and the MathBrush recognizer. Based on results from the user study, the user interface and other features within the application (besides the recognizer) seem to be well appreciated by a strong majority of the participants, and play an important role in improving the overall usability of the application.

Chapter 5

Conclusions and future work

The increasing role of computer algebra systems (CAS) in post-secondary mathematics education continues to largely impact the methods for how students study and solve mathematical problems. CAS offer the ability to remove much of the "manual labour" associated with performing calculations by hand, which has been long considered the traditional way of doing math. While some post-secondary math courses may require students to use CAS during labs or tutorials, typically a student's choice on when or what CAS to use is done on a purely voluntary basis. The growing accessibility of CAS via the internet has made modern CAS a more viable and appealing resource when completing math course assignments. Previous works have shown that learning a rigid syntax is one of the main limitations of students opting to use a computer algebra system. Natural language processing (NLP) is the main form of input for Wolfram|Alpha, which reduces the need for learning syntax, but can also cause some interpretation issues for large complex inputs. The MathBrush project makes use of pen recognition in its iOS and Windows applications, which allows a user to input their query in the form of drawn ink within the application. Pen recognition removes the need to learn any syntax, and attempts to bridge the gap between drawing on paper and interacting with CAS.

This thesis aimed to implement MathBrush as a web application, such that any user can access the application from an internet-enabled device and be able to comfortably communicate with computer algebra systems via pen-based input. This was accomplished in the form of three main stages:

1. Formative study: Semi-structured interviews were held with several post-secondary math students, with the overall goal of understanding the typical process a student

takes to complete an assignment. This included discussions on the tools/resources used, the benefits/limitations of those resources, and the conditions when certain resources are used.

2. Implementation: The front-end of the JavaScript-based MathBrush web application was built from scratch, and must have proper and efficient communication with the server-side recognizer and CAS wrappers.

3. User study: Semi-structured interviews were held with a subset of the participants from the formative study, with the main goal of acquiring user feedback for the MathBrush web application. This included discussing the strengths and weaknesses of the application and what future developments and improvements should be focused on.

The formative study revealed that most participants used a very similar process for completing their math assignments: review course material, do rough work on paper, verify solutions online, and finally write a final draft. Despite the growing number of software resources available to students, pencil/paper still plays a very dominant role in the rough work stage. Online resources (especially Wolfram|Alpha) were heavily used in the verification stage. It appears the context that pen recognition will be useful to students will also occur in the verification stage as a method of inputting their query to a computer algebra system. Several participants indicated they would not make use of pen recognition in the rough work stage since they like the "freedom" that comes with drawing on paper, and not having to worry about the accuracy of the recognition. In fact, the overwhelmingly largest concern discussed by participants in the context of pen recognition was the accuracy of recognizing their pen input. The complexity of a user's input also seemed to be a factor for when a participant would want to use pen recognition as a form of input. Several participants indicated that for simple queries they prefer typing (using natural language) into Wolfram|Alpha, but would likely make use of pen input for more complicated queries. In terms of design considerations, many participants encouraged adding the capability of linking the pen recognition with Wolfram|Alpha directly. Since they are already comfortable with and heavily use Wolfram|Alpha, the incorporation of pen recognition seemed like an ideal feature. The other major design consideration discussed in this study involved around creating a very simple user interface. Some current popular CAS (Maple in particular) was found to be overwhelming to use as a novice user for some participants, mainly because the software offered a far greater number of features than what they needed. With post-secondary mathematics students being the target user of the MathBrush web application, a simple and intuitive interface will be the most effective.

The implementation of the MathBrush web application presented its own set of challenges, mainly due to the architecture of the application. The recognition library (in the form of a compiled C++ shared library) was made a server-side functionality, as opposed to the iOS and Windows platform application where it resides on the client. This was mainly due to the lack of current web technologies which support running C++ libraries on the client. The rendering library was found to be incompatible with the JavaScript application, since the passing of a graphics context pointer between the JavaScript front-end and the server-side rendering library does not seem possible. MathJax was a substitute chosen as a means of displaying the recognized expressions and computed results, although the displayed mathematics cannot be rendered directly. Integrating CAS functionality into the MathBrush web application was done successfully, with the ability to pass the recognized expression to Maple and Sage (using the same wrappers from iOS and Windows versions), and have the computed results returned back to the application. Wolfram|Alpha compatibility was another implemented feature, which allows the user to pass the recognized expression directly to Wolfram|Alpha to display Wolfram's computed results. There were no libraries or frameworks used for managing the drawn ink within the front-end of the application, so these operations were manually implemented in JavaScript (i.e. adding strokes, deleting strokes, undo, etc.). In order to keep running instances of the MathBrush web application separate, clients were managed by passing generated keys between the client and server. This gives each client a unique key, which is passed alongside any data sent to the server, and maps directly to the server-side data corresponding to a particular client. Automatically computing the screen density of a device was found to be impossible, despite this being a heuristic used by the recognizer. An average value is hard-coded for this heuristic, although future user interface changes could allow a user to modify this value for increased accuracy.

The extent to which the MathBrush web application is considered "usable" was determined based on discussions during the application user study. Much of the discussions related back to concepts discussed in the first desgin study, such as: reliance on pencil/paper, pen recognition concerns, complexity of input, and software suggestions and openness to change. In terms of the user interface, most participants had mainly positive comments. Attempts to give the MathBrush web application interface a pencil/paper "feel" were appreciated by many of the participants as this aids the transition between writing on paper to writing on a tablet. However, it was also made clear that no piece of software (currently) would completely replace the role of pencil/paper, which will continue to be a consistently used tool when completing assignments. Overall the interface was found to be fairly intuitive to use for the majority of participants. Some participants indicated they would like there to be documentation or instructions for the application,

especially it relates to lesser-known features (such as the scratch out gesture). As for the pen recognition features of MathBrush, most participants were quick to point out that this is the application's biggest limitation. While the recognizer is by far the most complex (and most important) feature of the application, several participants had difficulties with their handwriting getting recognized. The main reasons for the recognizer's inaccuracies were due to: messy handwriting, touching symbols, drawing on a slant, and spacing issues. Adding the ability to "train" the recognizer based on a user's handwriting (similar to the iOS application) should also help in improving the recognizer. The complexity of a participant's query was determined to be a large factor when it came to how the participant wanted to interact with an application. For simpler queries, many participants preferred typing the input into an application like Wolfram|Alpha. For complex queries (i.e. containing many brackets and fractions), pen input was found to be the more popular method of input. A few participants suggested that the MathBrush web application should include alternative methods of input (such as typing) in order to accommodate various input preferences. Other suggestions to improve the MathBrush web application included:

- Adding box templates for drawing matcies and systems of linear equations in order to fix white spacing issues
- Adding graphing support (available in iOS app)
- Saving expressoins (available in iOS app)
- Improving the recognizer (most common suggestion)

Overall, the design and development of the MathBrush web application has been successful in offering a pen-based method of input to CAS from the web. Many of the strengths and limitations of existing math software were discussed in the formative study in order to develop a more useful application for post-secondary math students. By better understanding the common processes and HCI challenges associated with how math students complete their course assignments, the more pivotal a role the MathBrush web application can play for helping math students. The high level of openness for trying new applications that was expressed by students in the user study provides great motivation in continuing to design improved software to suit their needs.

5.1 Future work

In terms of improving the MathBrush web application as a whole, future work would put a great amount of focus towards improving the recognizer. With several of the participants

indicating the usability of the application is completely dependent on the accuracy of the recognition, some necessary improvements must be made. However, since this thesis treated the recognizer as a "black box", future work is instead more focused on addressing web-specific challenges. The ability to train a user's handwriting with the recognizer is a feature which could be added to the interface of the application, and does not require modifying the recognizer itself.

Adding rendering functionality to the application is a large area for improvement. Since it does not appear that the existing MathBrush rendering library is compatible with the MathBrush web architecture, rendering features may have to be written from scratch in JavaScript. The ability to render the recognized expressions is imperative towards correcting any inaccuracies. Continuing to add existing features from the iOS MathBrush application to the web application will also add breadth to its list of capabilities, such as plotting and saving/sharing expressions. Finally, by including typing as a method of input (similar to Wolfram|Alpha), students will have the freedom to choose their preferred means of inputting their queries, especially as this is influenced by the complexity of their input.

The inclusion of pen input for interacting with CAS provides students with an intuitive interface for entering queries. While learning syntax has been considered a disadvantage by students when learning how to use CAS, the ability to draw their input directly into a computer algebra system reduces this learning curve. Pen recognition, along with the accessibility of a web application, combine to create an ideal tool for math students. Computer algebra systems have revolutionized the area of mathematics for both students and researchers. The design and development of the MathBrush web application will hopefully play a significant and momentous role in the progression of computer algebra systems, and as a beneficial resource for students.

References

[1] AngularJS. https://angularjs.org/. Accessed: 2017-07-05.

[2] AngularJS Introduction. https://www.w3schools.com/angular/angular_intro.asp. Accessed: 2017-07-05.

[3] Express. https://expressjs.com/. Accessed: 2017-07-05.

[4] Infty project: Research project on mathematical information processing. http://www.inftyproject.org. Accessed: 2017-07-05.

[5] MathBrush: An intuitive Pen-Math System. www.scg.uwaterloo.ca/mathbrush/. Accessed: 2017-07-05.

[6] Mathjax. https://www.mathjax.org/. Accessed: 2017-07-05.

[7] The MyScript Equation recognizer from Vision Objects. http://www.visionobjects.com. Accessed: 2017-07-05.

[8] NodeJS. https://nodejs.org/en/about/. Accessed: 2017-07-05.

[9] Node.js Tutorial. https://www.w3schools.com/nodejs/. Accessed: 2017-07-05.

[10] Optical character recognition. http://www.pcmag.com/encyclopedia/term/48267/ocr. Accessed: 2017-07-05.

[11] Persisting Ink on the Web. https://msdn.microsoft.com/en-us/library/aa480684.aspx. Accessed: 2017-07-05.

[12] About Maplesoft. http://www.maplesoft.com/, 2017. Accessed: 2017-07-05.

[13] Sage. http://www.sagemath.org/, 2017. Accessed: 2017-07-05.

[14] Wolfram Mathematica. http://www.wolfram.com/mathematica/, 2017. Accessed: 2017-07-05.

[15] D. Adair and M. Jaeger. Making engineering mathematics more relevant using a computer algebra system. *International Journal of Engineering Education*, 30(1):199–209, 2014.

[16] Andrea Bunt, Michael Terry, and Edward Lank. Friend or Foe? Examining CAS use in Mathematics Research. *Proceedings of CHI 2009*, pages 229–238, 2009.

[17] Chantal Buteau, Neil Marshall, Daniel Jarvis, and Zsolt Lavicza. Integrating Computer Algebra Systems in Post-Secondary Mathematics Education: Preliminary Results of a Literature Review. *International Journal for Technology in Mathematics Education*, 17(2), 2010.

[18] Colin Flanagan. The Bresenham Line-Drawing Algorithm. https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html. Accessed: 2017-07-05.

[19] Scott Frees. Calling Native C++ DLLs from a Node.js Web App. https://nodeaddons.com/calling-native-c-dlls-from-a-node-js-web-app/. Accessed: 2017-07-05.

[20] J.J. LaViola Jr and R.C. Zeleznik. MathPad2: A system for the creation and exploration of Mathematical sketches. *ACM Transactions on Graphics. Special Issue: Proceedings of 2004 SIGGRAPH*, pages 432–440, 2004.

[21] Tsvetanka Kovacheva. Use of the Maple System in Math Tuition at Universities. *International Journal "Information Technologies and Knowledge"*, 1:363–368.

[22] G. Labahn, E. Lank, S. MacLean, M. Marzouk, and D. Tausky. MathBrush: A System for Doing Math on Pen-Based Devices. *Proceedings of The Eighth IAPR Workshop on Document Analysis Systems DAS*, pages 599–606, 2008.

[23] G. Labahn, E. Lank, M. Marzouk, A. Bunt, S. MacLean, and D. Tausky. MathBrush: A Case Study for Pen-based Interactive Mathematics. *Proceedings of the 5th Eurographics Symposium on Sketch-Based Interfaces and Modelling (SBIM 2008)*, 2008.

[24] Z. Lavicza. *The examination of Computer Algebra Systems (CAS) integration into university-level mathematics teaching.* PhD thesis, The University of Cambridge, Cambridge, UK, 2008.

[25] M. Lutovac and A. Lutovac. Maximal Payoff Strategy for Vickery Auction using Game Theory and Computer Algebra Systems. *Sinteza 2014 - Impact of the Internet on Business Activities in Serbia and Worldwide*, pages 66–70, 2014.

[26] S. MacLean and G. Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *International Journal of Document Analysis and Recognition*, 16(2):139–163, 2013.

[27] S. MacLean and G. Labahn. A Bayesian model for recognizing handwritten mathematical expressions. *Pattern Recognition*, 48:2433–2445, 2015.

[28] S. MacLean, G. Labahn, E. Lank, M. Marzouk, and D. Tausky. Grammar-based techniques for creating ground-truthed sketch corpora. *International Journal of Document Analysis and Recognition*, 14(1):65–74, 2011.

[29] Scott McLean. *Automated recognition of handwritten mathematics*. PhD thesis, University of Waterloo, 2014.

[30] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S. Vorketter, J. McCarron, and P DeMarco. *The Maple Advanced Programming Guide*. Springer-Verlag, 2003.

[31] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, S. Vorketter, J. McCarron, and P DeMarco. *The Maple Introductory Programming Guide*. Springer-Verlag, 2003.

[32] Robyn Pierce and Kaye Stacey. Observations on Students Responses to Learning in a CAS Environment. *Mathematics Education Research Journal*, 13:66–70, 2001.

[33] Mark Prosser. Solving mathematical problems on touch-based devices. Master's thesis, University of Waterloo, 2014.

[34] Tuan Salwani Salleh and Effandi Zakaria. Integrating Computer Algebra Systems (CAS) Into Intro Calculus Teaching and Learning at the University. *International Journal of Academic Research*, 3(2):397–401, 1 2011.

[35] Neil Morrell Soiffer. *The Design of a User Interface for Computer Algebra Systems*. PhD thesis, EECS Department, University of California, Berkeley, 4 1991.

[36] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge university press, 2013.