

# Survey of User Input Models for Mathematical Recognition: Keyboards, Mice, Tablets, Voice

*Lucy Zhang and Richard Fateman*

*Computer Science Division*

*University of California*

*Berkeley CA, 94720-1776*

## Abstract

The entry of mathematics into a computer system is important in some contexts: computer algebra systems, programming for scientific computing, educational training, and publishing. Different designs based on different user models may be appropriate. We survey some of the past approaches and suggest how new technology will (or will not) make these tasks easier.

## Introduction

Computer systems have become central tools in education, science, and technology for data storage, presentation, and communication. Text is routinely counted as “bytes” being stored digitally, images are being captured in bitmaps, and ideas are being exchanged through electronic media. While computers are able to ingest, interpret, and transcribe a vast amount of knowledge in textual form, they present barriers for humans to represent and express mathematics.

First of all, there exists a dichotomy between the appearance or presentation of math and the semantic meaning underlying the notation. Expressions appearing identical can have drastically different meanings in different contexts ( $dx$  as in the derivative with respect to  $x$  or  $dx$  as in  $d$  times  $x$ ). Formulas that have equivalent meanings can have different appearances ( $\ln x$  and  $\log_e x$ ). This presents an additional level of ambiguity that is not always recognized by computer scientists. The linguist studying natural language may view “Time flies like an arrow.” as interpretable in several ways, but one is still able to identify the separate words and provide a concise explanation of the possibilities. In mathematical utterances the meanings may require reference to the enclosing context for definitions and notational conventions in a manner that is inherently difficult.

To proceed on the path towards digitizing mathematical information, many groups will benefit from the ability to create and store electronic versions of scientific documents. These include publishers of journals, software publishers for document analysis, software

publishers for assessing document semantics, electronic libraries, and individual users who try to access the libraries<sup>4</sup>. In fact, we can extend the benefits of a good math input and representation system to a wider range of users. We will specifically assess three groups of users, K-8 students, high school students and undergraduates, and advanced researchers.

Being able to represent math in a computer really starts with the capability to enter math into such a system. There are several different models for math entry in existing programs which we will explore and evaluate. With the technology of tablet PC's on the rise, we explore the ability to incorporate this new technology into the existing models. And lastly, after examining the tradeoffs between different models, we will try to design a potential model which will combine features from the different existing models in an attempt to maximize each of their strengths.

## Classification of Input Models

Many mathematical computer systems that require math input provide the user with an interface that can be categorized into one of the four types of models: handwriting recognition, template and palettes, keyboard command entry, and typesetting systems such as TeX.

### *Handwriting Recognizers*

The handwritten recognition model provides a natural, nonrestrictive way of entering mathematics into a computer system. It does not require you to learn a new convention for conveying math. Also, Arvo [1] argues that handwriting math allows you to enter math without having to “mentally parse” the expression as you are writing. This seems to us not especially well supported, however.

Handwritten mathematics recognizers capable of handling simple expressions are fairly easy to build, and are often written as neat demonstration programs for new stylus-like input devices. In reality the evidence is that such systems do not

easily scale up and are limited in accuracy and robustness.

In a typical system the recognition process consists of two main phases, the symbol recognition phase and the structural parsing phase. Each phase contributes to a high error rate. Misidentification of only one symbol in ten would seem to be a low error rate at first blush, but consider that to correct one symbol requires (a) correct identification and positioning of the “rub-out” symbol (b) correct recognition and placement of the new symbol. Thus on average to enter ten symbols one will have to enter twelve or more. Some symbols have multiple strokes. This is irritating.

Misidentification of symbols is difficult to avoid given apparent similarities of symbols. Notice the overlap in characteristics of sloppily written different symbols as {l, 1, |, []} or {C, } or {α, , } or {p, }. This problem is compounded by the wide range of variations among users’ handwriting style; most handwriting recognizers need to be trained on a per-user basis.

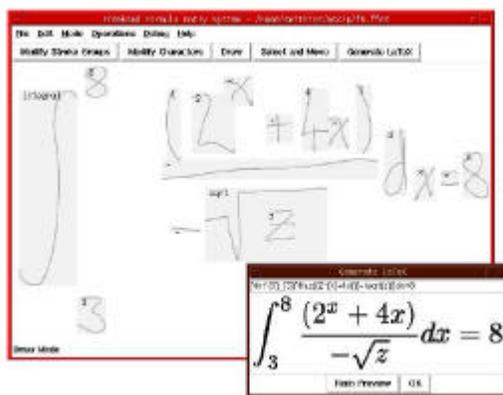
After character recognizer is finished, the math expression needs to be structurally parsed. (In fact, recognition and parsing may be profitably interwoven. Recognition of symbols may be dependent on expectations of a parser: is a horizontal line a divide bar or a minus sign?). Parsing of conventional mathematics is difficult because it tends to be ambiguous, even when given on a single line. Unambiguous “programming language” versions of mathematics are, contrary to the popular view among computer scientists, a small subset of the

richness of the usual mathematical notation. Add to this the need to disambiguate what amounts to subtle errors of misplacement (Is that a subscript?) and the possible number of parses grows substantially. It is not that the parser will take too long. It is that there are (perhaps exponentially many) alternative parses.

Building a handwritten math recognizer that can handle the needs of advanced mathematicians is a much more difficult task than building one which suffices for grade school students.

We must build in convenient correction technology. For example, allowing deletion of erroneous strokes, listing/choosing alternative interpretations, and learning from common mistakes help minimize the amount of extra work users must carry out in order to undo the mistakes of the recognizer. User feedback is crucial for the correction process and should be integrated with system learning. Since recognition errors in current prototypes (all we seem to have are prototypes!) are still fairly common, correction procedures should at least be more convenient than if the user were to just type in the expression into Emacs for a typesetting system like TeX.

Below is an illustration of a handwriting recognizer. It has a window area large enough for pen written strokes. There are methods to modify misplaced or wrongly written strokes. Most of the programs allow for selection of sub-expressions and moving the selected areas. The process of recognition in this case generates input for the typesetting language LaTeX which is then processed and displayed.



A prototype of a handwriting based math entry system<sup>10</sup>

### Templates and Palettes

A less natural and more constrained model of entering math is by means of interactive

construction using structured templates and palettes. These are windows resembling palettes which have buttons that represent predefined structures of basic math

operations such as fractions, exponents, integrals, square roots, summations, etc. The operands of these functions are left for the user to fill in using either keyboard commands or more buttons.



An example of a template from Mathematica

One advantage of a template model is this forced-entry guarantees well-formed expressions. Also, it allows the system to easily add new symbols and functions by just adding a new palette or extending an existing one. The disadvantages of this model pertain to ease of use. Manipulation of templates requires most users to switch between the keyboard and the mouse; although there may be keyboard shortcuts that can make certain commands more efficient, the number of entries is daunting. Also, finding the appropriate palette and the entry within the palettes in order to enter one expression may be frustrating for the user. For a touch-typist capable of 80 words per minute (400 characters/minute or more than 6 characters per second) typing `\Gamma` takes one second. Can you even find the  $\gamma$  symbol in the template above in one second?

Features can be incorporated in programs using the template model to compensate for the model's inherent weakness and emphasize on its strength. For example, to make navigation as efficient as possible, commonly used expressions of calculus such as integrals, derivatives, infinity sign, superscript and subscript for the limits should be organized on one palette. In fact, common commands that could be used in most any math expression may be grouped together.

These include superscript/subscript, fractions, square root, exponent, etc.

Constraining the users to enter only valid math expressions may be confusing for those who do not grasp what some well-formed expressions look like. Indeed, some intermediate stages that would eventually become well-formed cannot be entered this way: consider  $1/(a+b)$  with prefixes  $\{1, 1/, 1/(, 1/(a, 1/(a+, 1/(a+b)\}$  most of which are ill-formed, but valid prefixes.

Annotated assistance for what each palette button means and what operands are required may help, (certainly some of the Mathematica choices are obscure), but they still cannot make up for user expectations that do not match the template mode.

### Keyboard Command Inputs

Some (most) programs allow users to enter math using strictly the keyboard. Since the characters that appear in a math expression do not all exist on the keyboard, sequential combinations of keystrokes convey the special math-related symbols.

Users familiar with keyboard-based editors such as Emacs are likely to be accustomed to learning new keyboard commands and typing combination strokes, perhaps with command-completion so they may find such systems easy to operate. However, there are still some disadvantages to this model. Users must coordinate between typing commands and typing math. Unlike the typical text editor entry task where most of the time is spent typing in text and less time is used to issue Emacs commands, entering a math expression will generally require a higher percentage of positioning commands to manipulate the raw symbol input. In this kind of editing, we should point out, the typing results in a WYSIWYG display: typing something like `x y+z` will be seen as  $x^{y+z}$ . The intervening keyboard strokes may not be visible, may be difficult to select, delete or correct. This may require substantial learning to use effectively. Unlike the template model, using keyboard commands allow you to freely enter inputs that do not guarantee well-formed results.

Features that could accommodate some of the disadvantages of such models are provide a command that will undo previous inputs or commands to help you recover from mistakes. Also, allowing copy and paste of previous entered data can expedite similar or repeated entries of an expression.

### Typeset Systems: TeX input

Math typesetting languages such as the math mode in TeX are text-based formula description languages.

Advantages of typesetting languages provide the major advantage of being essentially formal programming mechanisms which are well-defined and (certainly in the case of Knuth's TeX) well-documented. They are robust in handling complicated math. There is also the added bonus of the expert appearance of the typical TeX display. Disadvantages can be the steep learning curve that is required in order to get started, and the long time to master the syntax of the language. "TeXperts" find it relatively easy to express themselves using the language. Our example from the previous section would be rendered as  $x^{y+z}$  in TeX.

A useful feature that is not usually incorporated in a typeset system, but certainly would be useful (see next section, though) would advantage of the typeset display of TeX by allowing you to copy and paste from them to reuse in other expressions.

### *Symbolic Computation Systems*

Most computer algebra systems (Maple, Mathematica, Macsyma, MuPad, Reduce,) themselves have their own user-input parser with thousands of separate operators, but are willing to change them to TeX for display purposes. Such systems can create new and more elaborate formulas by computation, selection, rearrangements and combinations, making some of the tedious and error-prone entry of large expressions unnecessary. In other ways such systems are no different from the categories above: most are typically driven by keyboard input. Seasoned computer programmers who constantly code in.

### **User Classification**

When considering models of interaction between people and mathematical computer systems, we should take into account the different purposes of usage and familiarity with the subject of the user. Therefore, we break down the types of users into 3 groups, and describe and analyze some features of a mathematical input model accordingly. The main objective for all groups is of course to enter math expression into a computer program.

#### *K-8 Math and Science students*

For this group of younger users, the main purpose for a mathematical computer system might be to

learn mathematics, including arithmetic or elementary geometry. Students' familiarity with the subject will be none to sufficient; mostly at the elementary level. At such a level, the students possess less knowledge of math which may lead to more frequent mistakes in input. Also, younger users may have illegible handwriting which makes handwritten inputs more prone to errors. Since K-8 students are not familiar with math, mistakes in recognition may confuse the user and may be overlooked. However, the less complicated mathematical expressions may make recognition simpler.

Taking into account the context of young and inexperienced users, we can build in features that accommodate their preferences. The main idea is to minimize the errors that could be introduced. Templates may be a good idea which will provide a guide for younger users who may often make mistakes in writing math. Younger users probably learn to input math via calculator before using a computer, so math input programs that model calculator input may be more receptive. Mouse-based operations may be easier for younger users (K-3) because pointing and clicking is a simpler skill than typing on a keyboard. An added benefit that a math computer system can provide at the high end of the age group is providing visualization of manipulation steps in elementary algebra.

#### *High School and College Math students*

Math students at the high school and college level already possess a decent fundamental understanding of advanced topics. The objective for this group of users is to help them visualize new concepts, aid in doing homework, and perhaps allow them to use computer systems to print out assignments for more legibility. College students in science areas are likely to have some exposure to computer algebra systems, most commonly Mathematica and Maple. Many calculus classes are now incorporating computer lab workshops in addition to regular classroom learning. The problem that arises is that students fail to find the introduction of these systems to be enriching; more likely they find learning how to use these computer algebra system an additional burden! We want to make math entry and manipulation on these program more natural and convenient.

One approach is to define a context for each individual specializing in a domain at any given time. For example, calculus students most likely need to use algebra, trigonometry, plotting, and

integrals, derivatives. By contrast, linear algebra students need to manipulate matrices.

An educational setting is fairly clear, and students already make use of this. The questions at the end of Chapter 10 use (mostly) the techniques in Chapter 10. Never the methods of Chapter 11, and rarely earlier chapters. The possibility exists to provide special templates for each chapter, so that the math input needed for entering expressions will be presented in some form (template, previously composed example expressions) so that users can copy expressions with minimal technical or typographic errors.

A handwriting-based formula recognizer can also benefit from context: a sloppy vertical line could be an integral sign, a part of an absolute value sign, a digit 1, or “evaluated at”. In a linear algebra setting a vertical line could be a mark on a determinant, or a matrix bracket.

#### *Advanced Mathematical Researchers*

Researchers and advanced students in the mathematics and science fields rely on mathematical computer systems to provide assistance in calculations and typesetting math formulas for inclusion in publications. The subject knowledge among this group of users is much deeper, and the demands on a math input system more substantial.

Users at this level will need to be able to enter fairly complicated expressions and they may use a wide range of conventional math symbols, additional alphabets such as Greek and additional obscure symbols<sup>6</sup>. TeX is often a standard supported by some publications, and popular among (some) authors. Alternatives would have to have some advantages in learning, efficiency, or some other metric. A handwriting recognizer will need to be extra robust, or trained on this larger class of symbols (perhaps on a per-task basis). Template input systems will need more palettes or extensible palettes to represent a wider variety of functions and symbols. Unlike the K-8 students, it is more likely that mistakes in recognition will be spotted soon; correction methods may need to be more sophisticated.

There is a need for recognizing long expressions with multiple levels of groupings which will require sufficient input surface. A possibility is to

introduce lines as guides for multi-level inputs such as fractions, sub/super scripts etc.

#### **Existing Programs**

There are a variety of systems developed in research institutions, some of which are cited in the references. For our purposes we first choose one handwriting-based program which has the major advantage of being packaged for portability, and hence there is at least a chance of our using it for our own experimentation.

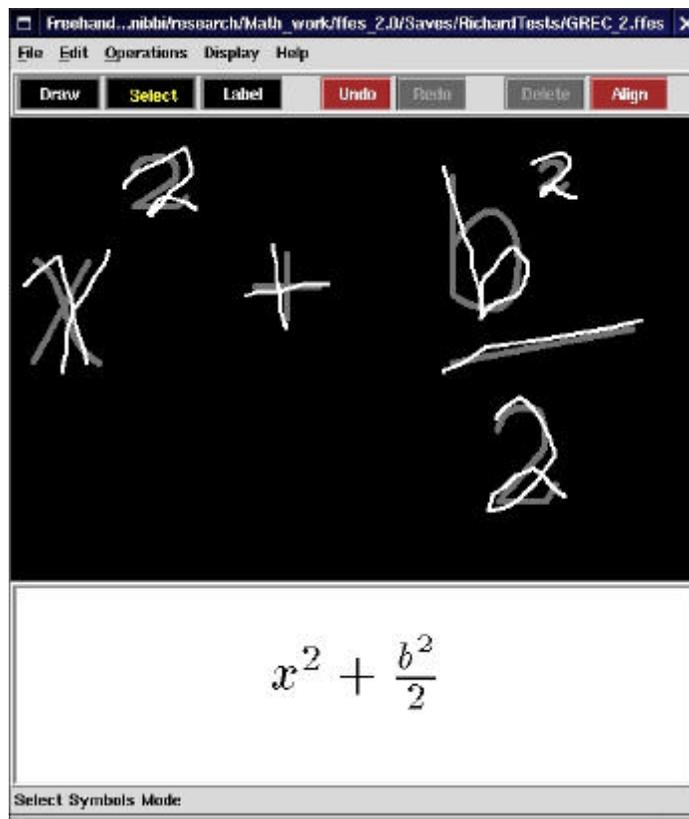
#### *FFES – Freehand Formula Entry System<sup>10</sup>*

FFES is a handwriting based equation editor. The user writes the expression in the window space provided, editing as needed using the undo option, selection, and the delete command. The recognized result is displayed using the TeX typesetting system.

The creators of FFES argue against text-based equation description languages such as TeX and formulation of expressions using structure templates. Their argument is that these require the user to mentally parse the expression and this is not normally part of the equation writing process<sup>1</sup>.

Although the authors claim that users don't need to mentally parse math expression when they are writing it down, it does not mean everything gets written from left to right. People often go back to add or match parentheses which create sub-expressions. Also, when writing a large fraction, we may draw the division line first and then go back to fill in the numerator and denominator. So, it could be argued that the user does think ahead and do some mental parsing even when writing freehand math on paper.

The authors admit that parsing the mathematical expression is still the slowest and least accurate part of the program<sup>1</sup>, hence reflecting the trade off between having a system that is easier and more natural to use versus one that is faster and has a higher rate of recognition accuracy.

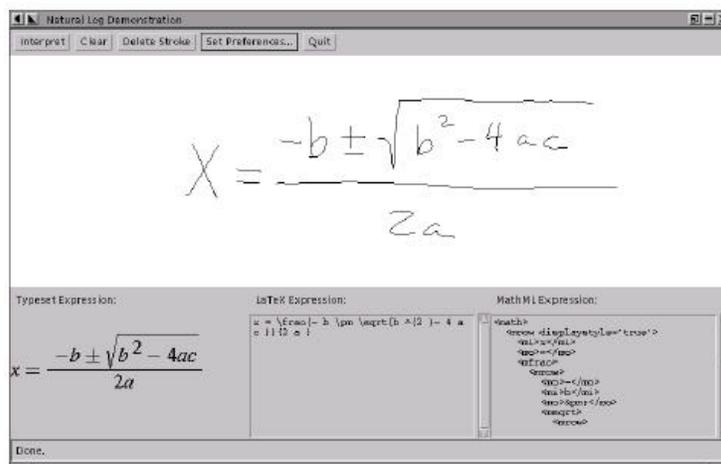


A screenshot of the FFES interface<sup>10</sup>

*Recognition of Handwritten Mathematical Expressions by Nicholas E. Matsakis*

In this thesis the author describes an on-line approach for converting handwritten mathematical expressions into an equivalent expression in a typesetting command language such as TEX or presentation MathML. This

system provides you a pen-based interface. You write with a stylus by in the space provided inside the program's window. There is a simple set of buttons at the top of the window. "Interpret" starts the processing of the handwritten input. "Clear" clears the screen. "Delete stroke" erases the last entered stroke. "Set Preferences" allows you to adjust parameters such as rejection threshold and combination weighting and displaying options.



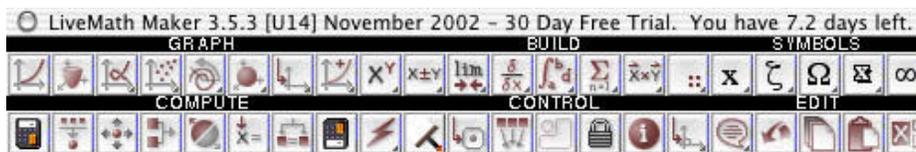
A screen shot of the prototype<sup>2</sup>

Some foreseeable problems with this model of input are that its error correction entails erasing strokes by scribbling back and forth over the stroke to delete. This may not work correctly if there are overlapping or clustered strokes because it's hard to precisely cross out a single stroke. The user may end up erasing wrong strokes which need to be re-written. One solution may be an extension of an existing feature of the system. Currently, the user is able to delete the last stroke written. A possible idea is to keep a history of strokes written. Then we can provide the user the option of deleting the  $i$ th stroke written. Maybe the number of history strokes to keep in the system can be set by the user depending on the complexity of the expression they will be entering.

The system also uses bounding boxes as a way of error detection. If the bounding box interpreted by the system is incorrect then users should be able to delete that bounding box. Also, there should be a user assisted way of indicating correct grouping of sub-expression. For example, being able to drag a box around sub-expression may suggest to the system to generate a bounding box around those sub-expressions.

### LiveMath

LiveMath is a template-based editor. Using the palette shown below along with highlighting sub-expressions, clicking, and dragging, you can control the entry of math.



A palette from LiveMath<sup>11</sup>

LiveMath allows you to build interactive math notebooks where you can manipulate the equations by selecting and dragging variables or expressions. The result of a manipulation will be evaluated and displayed. This program is suitable for elementary and high school students who want

to learn the steps involved when solving a math problem. It also provides a nice legible way of submit math assignments. Whereas most computer algebra systems only return the answer to the problem, LiveMath illustrates the steps of derivation.

$$\square a = x$$

$$\square I = \int_{-2a}^a (x^2 \sin[x]) dx$$

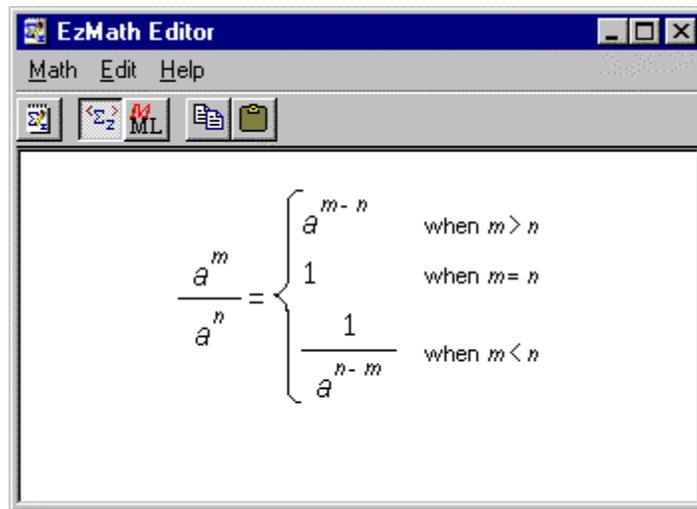
**Other CAS (Mathematica)**  
`Integrate[x^2 * Sin[x] {x, -Pi, Pi}]`

LiveMath output vs. Other CAS<sup>11</sup>

### EzMath

EzMath is a mathematical markup language used for embedding mathematical expressions in WebPages. It is inspired by the way expressions are spoken aloud. This provides us with some insight on how expression should be spoken.

EzMath focuses on the semantics of mathematical notation rather than how it appears on paper. The structure of each math expression when written using EzMath will be internally stored by the syntax of the markup language. EzMath was developed by Dave Raggett and Davy Batsalle.



A screen shot of EzMath Editor

The Ezmath for the expression in the figure above is:

$a^m$  over  $a^n$  = either  $a^{\{m-n\}}$  when  $m > n$   
 or 1 when  $m = n$  or  $1$  over  $a^{\{n-m\}}$  when  $m < n$

By pasting the above markup language into html, math will be display on a web page like the expression in the screenshot.

*MacKichan Software*

MacKichan is another template based math entry system which also gives the user of using only special keyboard strokes to input expressions.

Entering an integral using the mouse		
Step	Action	Result
1.	Click 	$\int$
2.	Click 	$\int \frac{\quad}{\quad}$
3.	Type x, click  , type 2	$\int \frac{x^2}{\quad}$
4.	Click in the denominator box, click 	$\int \frac{x^2}{\sqrt{\quad}}$
5.	Repeat step #3, click to the right	$\int \frac{x^2}{\sqrt{x^2}}$
6.	Type -9, click to the far right	$\int \frac{x^2}{\sqrt{x^2-9}}$
7.	Type dx	

$$\int \frac{x^2}{\sqrt{x^2-9}}$$

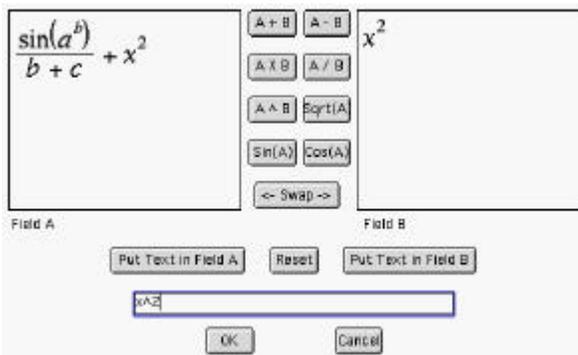
Using the mouse to enter an expression in MacKihan's math editor<sup>14</sup>

Entering an integral using the keyboard		
Step	Type	Result
1	Ctrl+i	$\int$
2	Ctrl+f	$\int \_$
3	x, Ctrl+up arrow, 2	$\int \frac{x^2}{}$
4	space	Moves the insertion point out of the exponent
5	Tab	Moves the insertion point to the denominator
6	Ctrl+r	$\int \frac{x^2}{\sqrt{}}$
7	Repeat steps 3 and 4	$\int \frac{x^2}{\sqrt{x^2}}$
8	-9	$\int \frac{x^2}{\sqrt{x^2-9}}$
9	space space	Moves the insertion point out of the radical, then the fraction
10	Dx	$\int \frac{x^2}{\sqrt{x^2-9}} dx$

Using the keyboard to enter an expression in MacKihan's math editor<sup>14</sup>

### Biscotti

This is a Java applet that has been an experimental prototype for input of mathematics both as a stand-alone "graphing calculation" and



was later re-targeted as an input form for Tiltu. Written by Eric Heien and Gifford Cheung at UC Berkeley, it provides the user with two panels and a collection of binary or unary operators to use to elaborate on the expressions. There is also another facility (not illustrated in this figure) to move a focus point into and around an existing expression. The 2-d display is computed using a program called Glyph-D by Ka-Ping Yee.

It takes in as input a LaTeX expression and produces an audio formatted output. Although it is in the wrong direction and the 1-d is verbal, it provides insight on the different ways math can be conveyed.

### Speech Recognition Input

A fairly unexplored area of mathematical input methods is using speech recognition. An inherent problem with using speech to input math is that math is not generally spoken, at least without an accompanying written presentation. It is difficult to verbalize lengthy and convoluted structured expressions without making mistakes. For

### AsteR – Audio System for Technical Readings<sup>13</sup>

AsTeR is a computer system for rendering technical documents in audio. Aster is spoken output, but nevertheless worthy of mention since it provides a mapping between 2-d and 1-d math.

example, sometimes people even stutter when reading a 10 digit phone. Also, it may be hard for user to force the habit of including certain phrases, i.e. saying “area code” before speaking the digits. While some voice recognition programs guess that any 10 decimal digit sequence is a phone number and should be “recognized” as (xxx)yyy-zzzz, this might in fact be incorrect. In the math recognition problem, we believe that analogous guesses will be less likely to work, even though they would be even more required.

In practice voice recognition of math requires symbol recognition probably beyond the current state of the art; it does not solve ambiguities already present in linear expressions (“A plus B over C plus D” versus “A plus B all over C plus D” etc.). Unintentional pauses and interjections of ‘ums and ahs’ are potentially more distressing in that in the math context they can change the positions of the next symbols, not just their identity.

Another idea we expect to be worth pursuing if voice recognition becomes faster and more accurate is to use speech as a correction method for handwriting input systems or to select from a template. This latter method is already used in some systems (e.g. Microsoft Office XP “voice command” mode). The mix of input channels seems to be an intriguing way of entering, confirming, or correcting handwriting.

### **Alternative Design Suggestions**

#### *Tapping and dragging*

For handwriting recognizers, it is essential to have a convenient way for you to modify or correct input. One design is to have you enter a symbol, and after the system recognizes it, it becomes a moveable box. At this point you can point, copy and drag it if it is misplaced structurally. After constructing an expression press “recognize expression” to process the input so far. The result will now appear in another moveable box and can therefore be placed and used as a sub-expression. This is a kind of customized template of pieces.

#### *Colored Ink*

If we can create a recognizer that distinguishes among colors, we can use color to specify groupings. i.e. **sin**  $x+y$  vs. **sin**  $x+y$ . This may help grouping ambiguities, and it may help with

semantic ambiguities such as the expression  $dx$  which can be:  $dx$  (the derivative) or  $dx$  (2 variables multiplied together). Instead of providing a palette of symbols and sub-expression, we can utilize a palette of various colors which the user can use to disambiguate structural uncertainty. Stylus “color” and other attributes such as pressure or buttons may be used.

#### *Handwriting Recognizer with Templates*

Given an expression to write, first write the symbols of the expression without worrying about the structure. The recognizer will process each symbol separately. After all needed symbols have been entered, point and drag them into a structured template. Template can be altered structurally to fit the structure of the expression. Then you can point and drag the symbols already created into the corresponding slots of the empty template. This separates the character recognition step and the structural parsing stage of the process. Also, manipulating the structure so it fits the expression you want to enter is entirely dependent on you: although the system can impose some force entry requirements. That is, specifying a fraction requires three parts: a numerator slot (a default fraction line) and a forced denominator slot. This helps enforce well-formed expressions as well as speedup of the structure organization step for the user. So, in a sense, this is a mixture of the template/palette model. Here instead of a palette we use a symbol recognizer to input the symbols. Palettes may be hard to navigate when writing the symbol we need is much easier. A more detailed description of this design is in a separate draft paper, “State Transition Chart for Handwriting & Template Math Entry System”.

#### *Speech Recognition using Groupings*

Math entry using speech recognition faces the obstacle of verbal mistakes by the user especially when the expression being spoken is long and complicated. To add to the problem, the way math is naturally verbalized can have ambiguous meanings. For example, when we say “x plus y over z”, this sentence conveys two different interpretations. It could mean  $(x + y)/z$  or  $x + y/z$ . One method of disambiguating the uncertainty is to introduce extra pauses, but this seems quite unreliable. More plausible is the use of keywords or voice commands added to the spoken language of mathematics. For example, to signify that everything spoken so far should be grouped, the

user must say the keyword “all” to instruct the grouping. Therefore, the first expression,  $(x + y)/z$ , would be spoken as “x plus y all over z”. If “all” is not used, then the previous standalone sub-expression is used as in the case for the second expression in the above example. Another possibility is to vocalize the second form as x and plus y over z. Both “and” and “all” could be repeated as kinds of traversals up an algebraic tree representing the expression.

As you can see, the introduction of keywords creates an even longer and complex spoken language for the user to learn. Though the keywords will help the speech recognizer parse the spoken words, it leads to a more error prone manner of vocalization for the user.

Another solution to this is grouping sub-expressions, naming the sub-expressions, and building bigger expressions with these labeled sub-expressions. Using spoken commands such as “group” and “end group” we can instruct the speech recognizer to group certain sub-expressions. After grouping a sub-expression, the user can voice the command “name as” to assign a name to the sub-expression such as “expression 1”. The advantages of this method are:

- Since we group and save sub-expressions, they are ready for reuse in future expressions.
- By breaking down the construction of a complex expression into smaller sub-expressions, we put fewer burdens on the speech recognizer which will now be required to handle simpler structures.

Using grouping commands allows the users to build up a complex expression in a straightforward and organized fashion.

So now to input the expression  $(x + y)/z$ , we say “x plus y group, name as expression 1, expression 1 over z”. To input the expression  $x + y/z$  would still be “x plus y over z”, but we can group it as a sub-expression by saying “name as expression 2”. This allows the usage of this expression in later expressions. For example, if we need to input the expression,  $\sqrt{x + y/z}$ , we say “square root of expression 2.”

Yet another approach would be to have the computer display the possible choices in an indexed list. To continue, you would have to choose which interpretation is correct.

## Alternative Uses of a Tablet for Math

A simple usage of a tablet is to input math with a pen and save it as an image. This ability is helpful in many applications. Oftentimes users may need to convey math to other (humans) through an electronic device such as email or instant messenger. These situations do not require any internal representation of the math. This can provide a convenient way for people to discuss math over the internet.

## Tablets vs. Mice

Operating a handwriting math recognition system by using a tablet and pen is a different experience than using a mouse. First off, tablets often have touch sensitivity up to 512 levels of pressure sensitivity. Also, some tablets have surface coating to diminish glare and to simulate paper-like feel<sup>9</sup>. In a study done to evaluate input devices, usage of a stylus was compared to that of a mouse. The results revealed that a stylus was slower in dragging than a mouse<sup>7</sup>. The reason for this may be the kind of “automatic transmission” in the mouse software that is rate sensitive, and will change gears for long moves. However, with a stylus, the movement of the hand from point A to point B is the same distance as the migration of the pointer.

When dragging a mouse, the mouse button must be depressed. With a stylus, the user must maintain pressure on the stylus. Pressing a button is a discrete operation. It is either pressed down or not. However, applying pressure on a tablet is a “continuous” movement. If the pressure falls below a threshold, the dragged object may be dropped. This is similar to pressing a mouse button vs. tabbing on the touch pad on a laptop. So, it is logical that dragging with a stylus is more prone to dropping errors.

Based on the conclusions of this study, designs of tablet systems for math may benefit from having multiple modes: distinguish between pointing/dragging or other selection commands in one mode, and writing in another. Human factors experiments may be the best way to identify winning strategies.

## Acknowledgments

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Lucy Zhang was supported in part by an undergraduate research grant from the College of Engineering, UC Berkeley.

## References

- [1] James Arvo, Kevin Novins, Steve Smithies. A Handwriting-Based Equation Editor.  
<http://www.cs.queensu.ca/drl/ffes>
- [2] Nicholas E. Matsakis. "Recognition of Handwritten Mathematical Expressions," Department of Electrical Engineering and Computer Science, MIT, 1999  
<http://www.ai.mit.edu/people/viola/research/publications/matsakis-MS-99.pdf>
- [3] Kam-Fai Chan and Dit-Yan Yeung. Mathematical Expression Recognition: A Survey. Technical Report HKUST-CS99-04, April 1999.
- [4] Richard Fateman. More Versatile Scientific Documents. University of California, Berkeley.  
<http://www.cs.berkeley.edu/~fateman/MVSD.html>
- [5] Zhao Xuejun, Liu Xinyu, Zheng Shengling, Pan Baochang and Yuan Y. Tang. Online Recognition of Handwritten Mathematical Symbols. ICDAR 97
- [6] An Automated Conversion of Structured Documents into SGML. Distributed Object Computation Testbed, Technical Report.
- [7] Scott MacKenzie, Abigail Sellen, and William Buxton (1991). A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems*, pp. 161-166. New York: ACM.
- [9] Wacom Americas.  
<http://www.wacom.com/lcdtablets/index.cfm>
- [10] Free Formula Entry System.  
<http://www.cs.queensu.ca/drl/ffes/>
- [11] LiveMath. <http://www.livemath.com>.
- [12] EzMath.  
<http://www.w3.org/People/Raggett/EzMath/>
- [13] AsTeR.  
<http://www.cs.cornell.edu/Info/People/raman/aster/demo.html>
- [14] MacKichan Software Inc.  
<http://www.mackichan.com>
- [15] J-Y Toumit, S. Garcia-Salicetti, H. Emptoz, "A Hierarchical and Recursive Model of Mathematical Expressions for Automatic Reading of Mathematical Documents," ICDAR 99 119-122.
- [16] R.Fukuda, Sou I, F. Tamari, X. Ming, M. Suzuki, "A Technique of Mathematical Expression Structure Analysis for the Handwriting Input System," ICDAR 99, 131-134.
- [17] TILU: Table of Integrals Look-up.  
<http://torte.cs.berkeley.edu:8010/tilu>
- [18] C. Faure, References on math formula recognition  
<http://www.tsi.enst.fr/~cfaure/math.html>
- [19] S. Lavirotte and L Pottier, Optical Formula Recognition,  
<http://www-sop.inria.fr/cafe/Stephane.Lavirotte/Ofr/root.html>  
also, ICDAR 97