Handwriting + Speech for Computer Entry of Mathematics
Work in Progress

Richard Fateman
Computer Science Division
University of California
Berkeley CA, 94720-1776

Abstract

The entry of mathematics into a computer system is important in a variety of contexts: educational training and testing, publishing and communication of mathematical results, use of conventional notation for scientific programming. Numerous keyboard and mouse-activated methods have been implemented and used in various systems; in their most ambitious form these have been in computer algebra systems. Handwriting has been repeatedly proposed and demonstrated (since at least 1968). Yet the use of handwriting alone has never put into production because of very high error rates. We now believe that a better prospect may emerge from using a combination of handwriting and voice.

Introduction

In an earlier paper, Zhang and Fateman [20] surveyed some of the existing methods for entry of mathematics into a computer system. Our conclusion was that additional methods, in particular combining voice and handwriting, deserve some attention for some potential user groups. When we realized that the discussion of this dual-mode was taking too much space in that paper and yet needed more discussion, we broke the paper into separate parts. This is part two.

In order to be somewhat self-contained, we review just a few specific relevant pieces of technology discussed in the earlier paper. The first of these is FFES, freehand formula entry system which, so far as can be seen from the open literature, is at least as good as the numerous earlier (or later) systems (but see also the promising non-open source Infty system [16], and the system "Natural Log" by N. Matsakis [2].) FFES was written principally by James Arvo at CalTech, but later adopted and refined by Dorothea Blostein and students at Queens University (Canada). AsTeR is a program that does a partial inverse of what we propose, that is, it speaks mathematics from a TeX representation. While not directly usable by us, it shows that there is at least one plausible mapping between TeX and speech.

FFES – Freehand Formula Entry System[10]

FFES is a handwriting-based mathematical formula editor originally written principally by Steve Smithies in 1998-99 at Univ. of Otago, under the supervision of K. Novins. More recently it has been revised by R. Zanibbi at Queens Univ. under the supervision of D. Blostein. Zanibbi in particular rewrote the parser, DRACULAE to be more efficient and accurate.

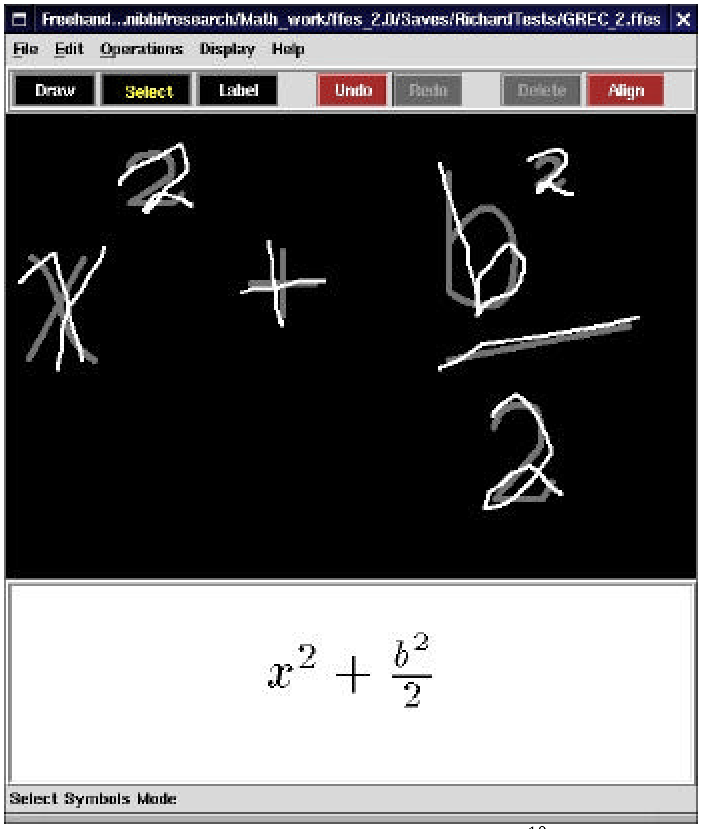
One component of FFES is CIT, a recognizer for handwritten symbols, written principally by J. Arvo at Caltech.

The user writes the expression in the window space provided, editing as needed using options for "undo", selection, and deletion. 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[1].

We don't actually agree, since for a complex expression a user will think ahead or else be forced to backtrack to insert parentheses or extend divide bars etc. Such insertions are possible in FFES.

The FFES authors admit that parsing the mathematical expression is still the slowest and least accurate part of the program[1], hence FFES reflects a 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. We suspect that some of the problem is the need to solve an inherently ambiguous problem by multiple pattern matching. What is a vertical stroke? Part of a letter K, the number one, a symbol? (Zanibbi's work seems to have sped up the parsing, as it happens; now display is apparently expensive.)


Freehand...nibbi/research/Math_work/ffes_2.0/Saves/RichardTests/GREC_2.ffes
File Edit Operations Display Help
Draw Select Label Undo Redo Delete Align
Select Symbols Mode

$$x^2 + \frac{b^2}{2}$$

A screenshot of the FFES interface[10]

Similar in many respects to FFES are three other formula-input systems, Infty: an elaborate system which seems to have begun life as a static "OCR" math recognition environment, Matsakis' "Natural Log", and Ernesto Tapia's JMathNotes, a Java program. FFES and JMathNotes are open-source.

Each of these can be downloaded from the Internet for experimentation, although Infty requires a (free, at the moment) license.

A kind different kind of technology we feel is worth highlighting is represented by AsTeR.

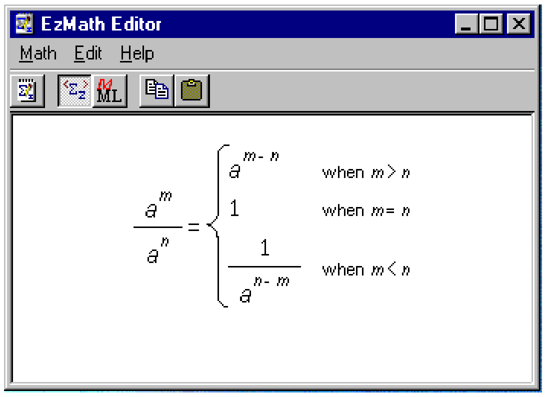
AsteR – Audio System for Technical Readings[13]

AsTeR is T.V Raman's computer system for rendering technical documents in audio. By analogy with Text To Speech programs (TTS) Aster provides spoken mathematics *output*. Even though we are primarily interested in input, it is nevertheless worthy of mention since it provides a mapping between 2-d math expressed in TeX and 1-d spoken math. It takes in as input a LaTeX expression and produces an audio formatted output. It provides insight in the different ways math can be conveyed. (It is also significant for those who are, like Raman, vision-impaired, and need to hear mathematics.) Examples of AsTeR output, transcribed, are in an appendix. The current state of the art in TTS makes it plausible to provide on web pages, audible HTML, which with a suitable browser, can simulate AsTeR. We have written such a program in two stages. The first returns a speech xml string when given an expression written as a prefix tree in Lisp. This result can then be fed into a second stage which reads this aloud, perhaps via a web-enabled speech engine.

Speech Recognition Input

A fairly unexplored area of mathematical input methods is using speech recognition, except as a tool for visually handicapped humans. An inherent problem with using speech to describe math is that math is not generally spoken, and so we are lacking familiar models. AsTeR provides a model, but is not especially appealing among sighted people who expect an accompanying written presentation. One of the successes of AsTeR, is that mathematically-aware but otherwise untrained humans can often understand AsTeR's spoken mathematics, although they would have difficulty producing similarly nuanced AsTeR-equivalent speech: humans tend to speak mathematics only under duress, and would in any case have difficulty learning and applying the mathematical rules for fine control of pauses and pitch used by AsTeR. Alternatives for spoken mathematics have been discussed by Stevens[] and Chang[].

Raggett's EzMath program [12] has a notation which is "inspired by how expressions are spoken aloud together with a few abbreviations for conciseness." For example, `"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"` produces the following image:



$$\frac{a^m}{a^n} = \begin{cases} a^{m-n} & \text{when } m > n \\ 1 & \text{when } m = n \\ \frac{1}{a^{n-m}} & \text{when } m < n \end{cases}$$

It is possible, presumably, to insert this into a program this as spoken input, but you would have to decide how to pronounce each component. For example is a^m "Ay to the power em" or "Ay caret em" or "Lower-case ay up-arrow lower-case em" etc. Caret would likely be recognized as "carrot." The grammar could be set up to allow any or all of these variations. To use the existing Microsoft speech software much of the contextual grammatical conventions that work so well in ordinary language must be reconsidered. Extensions may be possible, but in some cases it appears more likely that reversal of some standard choices would be needed to include mathematics. Current speech programs can be given a rather different language model to provide support for mathematics, and we have experimented with grammars to provide higher probabilities for more likely speech utterances including math operators, (plus, times, squared), isolated characters from Latin and Greek alphabets, etc.

The default speech recognizer knows about numbers, but makes rather different conventional use of them than would be done in technical writing. For example, it sometimes parses a stream of 10 digits as a telephone with area code.
Our experiments are just beginning, to see how learning and using conventions for mathematics may be critical factors in success, along with the quality of the microphone, the uniformity of the speaker's voice, the ambient noise, the level of training, and the quality of the speech program. At the current quality of recognition technology, we guess it would be unreasonable to expect first-time casual users – say students taking a college entrance examination – dropping in to an untrained environment, to find this convenient.

In the math recognition problem, a system may p have to switch between grammars or "probable-word" contexts, while recognizing non-math and math. In our preliminary tests, mathematics words are usually not the primary returned object because they are unusual, or they are placed in unexpected contexts. "Sum" becomes "some" or worse. "Sin" becomes "sign" or worse. If we wish to recommend to the recognizer a collection of phrases (like "sine of", "derivative with respect to", "to the power") it is more likely to make satisfactory discriminations. If the number of phrases can be grown to several thousand, (an experiment in progress now) so that the roughly 1,200 names of plausible MathML objects can be included, we have an interesting facility. For those unfamiliar with speech recognition technology, it may come as a surprise that word dictionaries and grammars play such an important role in quality recognition. In fact, recognizers benefit from more connected speech, and have difficulty with isolated context-free words. The important context may need to come from a grammar for mathematics utterances that would accept "open square bracket x plus y close" and produce [x+y]; and "open paren x plus y close" and produce (x+y). The fact that the same word, namely "close" is used for two different symbols is not necessarily a problem.
Once the stream of words is recognized, using whatever techniques can be reliably developed, one could try using a grammar similar to Raggett's EzMath as a guide to speech. A close examination of EzMath suggests it would have to be extended, perhaps in a major effort, beyond the current version of EzMath which seems to be brittle in the face of syntax errors.

In practice, voice recognition of math *solely from voice*, even in the event that all the symbols are correctly identified, 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.

A grammar and parser that we developed for an online lookup table (for integrals) is much more forgiving, and may have better prospects for success. (It will, for example parse ab sin x as "a*b*sin(x)". It will provide several parses for an expression which is ambiguous, and ask the user for assistance.

Alternative Design Suggestions

Multimodal

The mix of input channels seems to be an intriguing way of entering, confirming, or correcting handwriting. Mixed input in the common literature today usually means voice commands as an alternative to pushing buttons. Speech can be particularly effective if the options open for consideration are severely restricted, say to one of "File, Edit, View, Help".

After some investigation, we believe that a simple display/laptop mounted microphone is inadequate for serious use. The software is too sensitive to noisy input or varying volume and so the head-mount microphone is essentially necessary. An auxiliary WACOM tablet can be attached to a desktop workstation, or a Tablet PC can be used for writing. Writing proficiently with a mouse is difficult and slow.

In an education context it seems that students are already equipped with quality earphone headsets attached to music systems, but not microphones.

We are not in a position to evaluate the seriousness of the need for new (though rather inexpensive) hardware. (Computers themselves have gotten sufficiently fast that it is probably impossible to purchase a new desktop or laptop computer than lacks sufficient "horsepower" to run current speech recognition software. Windows XP includes as standard, speech tools.)

How can we use speech in mathematics? We proceed from the simplest to the more ambitious.

Better symbol identification

Independent of a more effective math language model, a simple objective for multimode input is to use speech as a correction method for *symbols*: that is, to select by voice the best choice for mapping a written symbol to one of a list of known symbols from a visual (or perhaps spoken) menu. The assumption here is that the user is dictating symbols or short phrases, and has a very primitive language model (like Microsoft's "spelling mode"). Thus writing the Greek letter μ and saying "myu". Because this may not be insufficiently distinct from "u" pronounced "yu" the computer may whisper into your ear "Did you say Greek mu ?" and your response may be to speak "Yes".

Alternatively a more elaborate message might be whispered or displayed as a menu: "Do you mean 1. 'Greek mu' or 2. Roman 'u' ?" And your spoken answer may be "one".

The usual default of displaying only the most likely symbol (and suppressing other possibilities unless asked) makes the recognizer seem rather brain-damaged when it is wrong, yet only modestly very clever when it is right. A skilled user presumably knows how to ask for a list of possible corrections to a wrong result, but the naïve user just wonders how the computer could have gotten the recognition so wrong. In effect the spoken input should improve the accuracy as well as the confidence. As an example, the Infty Project [] editor never recognizes a vertical stroke (|) as in |x|. It often recognizes such a written character as the digit 1. However, tapping once on that character replaces it by the next most-likely interpretation, which is the vertical stroke. Similarly there is only one symbol recognized for "O" and if you want a zero, you tap once. As a final feature worth noting in Infty, it will refuse to recognize some letters without help. One of them is the letter P, indistinguishable without baseline information, from p. A solution is to write the former as P.

Enhancing token separation

Consider what we can do to mimic the presentation of mathematics as traditionally done in the classroom. In this scenario the presenter speaks while writing. There are probably several motivation for a lecturer to speak while writing mathematics. It may improve the accuracy of students taking notes. It may enliven the presentation. It certainly fills the gaping silence (except for chalk noises) that would otherwise prevail. For the computer environment, writing the "word" a+e while saying "a plus e" may steer the handwriting system away from recognizing "ate". Seeing a+e may steer the speech recognizer into the correct path so that it would not hear (for example) "ape loss see," or worse. Using an example from Matsakis, consider which could either mean $1<x$ or *KX*. The re-parsing of the left-most stroke to be the digit one or part of a letter K (repeatedly reconsidered as other strokes are added) is an interesting feature of Matsaki's work.

Disambiguating operator types

Writing sin ωx while saying "sign of omega eks" illustrates not only the characters and tokens, but also suggest that sin is an operator. The "of" is not visible, but distinguishes function application f(a+b) read "eff of ay plus bee" from multiplication a(a+b) read "ay times ay plus bee" or "ay ay plus bee". Many math users are unaware of such ambiguities in their everyday notation, and computer users are similarly unaware because the designers of the languages they use remove these problems. The designers insist on adding operators like a*(a+b), or in the case of Mathematica, requiring f[a+b]. Computer users are inured to capricious and inconsistent requirements that such syntax rules are hardly noticed.

Enhancing structural identification

In addition to symbol and operator disambiguation, structural information can be supplied: saying A plus B over C plus D while writing

will distinguish the utterance from A+B/(C+D) or A+(B/C)+D.

Without belaboring this point we think that combined, a tablet and stylus used for free-form input will be significantly more convenient for the skilled user, and perhaps the naïve user, than otherwise possible with the tablet alone.

It may be necessary to have a multiplicity of possibilities mapped into common forms. "A over B" and "A divided by B" should be alternatives. Phrases like "eks squared" or "eks to the third power" may be acceptable. A grammar of forms and alternatives should be part of an extensible dictionary of forms. The guidance for contents could include forms from TeX, AsTeR, OpenMath, MathML. There is a design process for a standard XML format for speech grammars at W3C:
http://www.w3.org/TR/speech-grammar/

An alternative or supplementary approach through stochastic language models may also be appropriate. This is discussed (for example) by W3C in http://www.w3.org/TR/ngram-spec/ . Making use of either of these approaches would require a corpus of "people speaking mathematics" for synthesis and training.

Blank space, pauses, vocal directives

In terms of grammars as suggested above, one must come up with new markers for certain mathematical typesetting objectives. It is impossible to write "blank spaces" – at least as blank spaces. One could use special marks for writing them or *one could use speech* to assert information on blank space. That is, one could say "equation number 3.4 on right" or even "right tab to right margin equation number 3.4". The keyboard input might have "tab" characters, but the handwritten material would have "nothing" and then 3.4. Other kinds of grouping information may also be verbally conveyed. To explain this we use an analogy of a student standing at the blackboard holding a chalk and an eraser and attempting to write something that the instructor is dictating. The instructor can have "significant pauses," verbal commands such as "big open bracket", or "ALL over the expression…"., to signify that everything spoken so far should be grouped. Both "and" and "all" could be repeated as kinds of traversals up an algebraic tree representing the expression. The most useful verbal clue may be saying "no" to alternatives as they are presented. A student who knows the context is going to do a better job of understanding the voice input.

The introduction of keywords creates a longer and more complex spoken language for the user to learn. If we need voice to work even in isolation from handwriting, we could provide controls to add more structure as, for example, grouping and then naming the sub-expressions. Of course a voice recognizer could be used as a substitute keyboard, and so any linear ascii encoding for mathematics could be used as a proof of concept!

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:

- The groups and saved sub-expressions are ready for re-use immediately
- By breaking down the construction of a complex expression into named smaller sub-expressions, we put fewer burdens on the speech/handwriting recognizer which need be required to handle simpler structures.

This can be appealing only if naming is very convenient and users become facile with this concept, which is fortunately familiar to many users of graphical programs where structures are similarly grouped to build compound shapes.

As an example, to enter 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 use 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 named subexpression choices in an indexed menu or list. Expressions may be enclosed in small display packages using what are now fairly commonplace typesetting features. Variable sized characters, "stretchable" divide bars etc. can be provided. Some subexpressions may be parameterized. That is, they are more like templates. After placing such an expression in a target, the parameters may be filled in by pointing/writing and speaking.

Some Other Considerations

Feedback

It is possible that one could devise techniques to make use of audio feedback, say by reading back the handwritten material and thereby teach the user how to make correct statements in some way, or warn the user that something is uncertain. It seems this is an area for some experimentation. It is unlikely that choosing from an audio menu "if you want to close this parenthesis now, say *close*" is the way to go. The sophistication of possible math-reading programs like AsTeR [13] provides some context for building feedback.

Colored Ink

Tablet stylus systems generally allow you to write in different colors. A recognizer that distinguishes among colors can specify groupings. i.e. sin x+y vs. sin x+y. This may help syntactic grouping ambiguities, and it may help with semantic ambiguities such as the expression dx which can be: dx (the differential of x in an integral) or dx (2 variables multiplied together). Mathematica uses a keyboard entry variant ® to indicate the differential marker. Such subtleties may be hard to distinguish verbally or by handwriting nuances, so it may help to utilize a palette of various colors to disambiguate structural distinctions. Stylus attributes such as pressure, tilt, or buttons are at the designers' disposal, though it is not obvious that the ordinary user would become skilled in using these variations.

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, say by Arvo's symbol recognizer [1] point and drag them into pre-existing structured templates. Templates can be selected from a menu or perhaps 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, and in fact disambiguating horizontal lines becomes unnecessary. The divide bar can only come from a template.

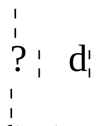
Templates provide the first example of promotion of well-formed expressions through *forced entry* requirements. That is, specifying a fraction requires three parts: a numerator slot (a default fraction line) and a forced denominator slot. The user must provide two of them. This helps enforce well-formed expressions as well as a potential speed up 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 or construct the palette. A more detailed description of this design is in a separate draft paper, "State Transition Chart for Handwriting & Template Math Entry System".

Alternative Uses of a Tablet for Math

A simple usage of a tablet is to write math with a stylus and save it as an image. This note-taking ability is helpful if an immediate electronic transcript, including diagrams and formulas is useful.

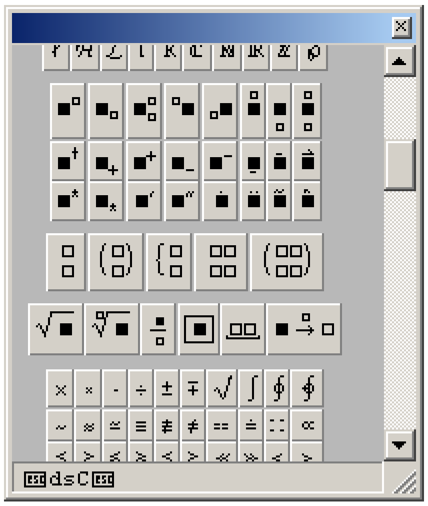
The application could be as simple as sending an email attachment of a formula. Oddly, most people talking about tablet recognition believe that temporal data (strokes, directions, delays) are in some sense important. The reality of formula recognition seems to deny this. We rarely benefit from watching someone write a formula; it must be recognizable from the ink that remains on the page. Only occasionally when (say) terms in a matrix are filled up in a particular order, or elements of a derivation are traced by "canceling terms" is the temporal dimension used. Even so, most mathematicians would re-write the formula after a few transformations. Certainly this simple act of encoding a written page as a static diagram to be emailed can solve a good part of informal electronic math communication.

It need not be this way. That is, temporal information could be used through feedback and prompting. An entry of a symbol (say the integral sign $\int$) could provide, as feedback, a picture requiring or suggesting input of additional material which could then be handwritten in place. That is, something similar to this:



indicating that plausible further symbols would start (but not necessarily remain within) one of the indicated gray areas. This technique of forced or constrained input works effectively only if the users' intent is effectively anticipated. The user would not be happy if the intended next step were to modify the symbol to a contour integral, $\oint$ which is not one of the choices. Similar forced choices include the supply of a matching bracketing symbols, so that writing a "[" displays a matching "]" just as a "d" appeared in the integral form.

How many such templates are available? The Mathematica computer algebra system has an extensive collection of such possible (but sometimes only remotely plausible) arrangements. A palette, which in this case also includes an easy way of entering some of the symbols not on the keyboard, looks like this:



(note that this is only part of a much larger scrolling palette, and only one of several palettes in that program.

Returning to our handwriting mode, note that there are tools available in the Microsoft SDK for the Tablet PC which suggest that handwriting cues ("factoids") and grammars could be offered to the recognition program. Thus the requirement that (say) a "+" operator is NOT likely to occur as a superscript or just following another "+" may be helpful.

A possible variant on matching in palettes is inspired by the handwriting input (as is offered on the Tablet PC operating system). You write in the panel, typically docked at the bottom of the screen. After a pause in writing, or perhaps when prompted by clicking on "insert" the material written on the panel is inserted into the chosen application. There may in fact be several panels or sections of this panel. For instance, a section in which writing would produce capital letters, another section for lower case, and one for numerical input. We can suggest that for each palette a panel can be used to recognize only elements on the palette. Thus the Greek palette would have no difficulty with recognizing a handwritten .

Tablets = Display?

The tablet PC makes it possible to display on the writing surface, as opposed to an opaque add-on tablet [9]. This saves space but is not entirely an advantage. It takes some practice, since one's hand can obscure what is displayed below or near the stylus. A two-surface arrangement with a horizontal tablet and a separate display gives an unimpeded view of the screen, but requires some adaptation to be comfortable. In either case, writing on a slick surface with a plastic-pointed pen is perhaps the greatest initial displacement from expectations. It is also worth noting that the "inking" program for Office tools not running on a Tablet seems far less accommodating than some other tablet management programs which take advantage of pressure and speed to make your handwriting look more like handwriting. We would not be surprised if this difference is easily remedied.

Experimentation with Speech

It is fairly clear (July, 2003) that using the default speech application, especially with a cheap microphone, will not allow for serious input of mathematics. Saying "a plus b" has a low probability of getting close. Fortunately a quality headset/microphone (August, 2003: Sony DR-260) does much better: With considerable training, with four trials I can get in succession "a+B. eight plus B a+B. A.+B." The extra punctuation and capitalization is annoying but perhaps repairable. On the other hand "a or b" is, so far as I can tell, impossible to say acceptably. It comes out "a war be". The word "or" is not the problem, since it easily recognizes "wet or dry." "True or false" is easy, "false or true" is harder. Additional training of (my) voice beyond about 15 minutes does not seem to help much, but 15 minutes of training may suffice.

It seems that provision of a good microphone is an absolute requirement, even on a Tablet PC (We tested, in May, 2004, a second generation Tablet PC, the Toshiba M205-S810. It does not overcome the need for a separate microphone.) Providing some voice training opportunity for a naïve user would also be a strong recommendation, given the current level of speech technology and the high accuracy that cannot otherwise be attained.

We are exploring a grammar-based language model for spoken mathematics, which may be constructed using the Microsoft speech tools. This in principle seems to provide a method for defining well-formed recognition segments and the possibility of associated semantics. A simple effort for speaking mostly context-free letters and symbols seems like a retreat from the phrase-based recognition of the default speech applications, and there are other hazards of home-built grammars (including the inability of the current (SDK 5.1 speech application) to allow for alternative recognition. We are working to find alternatives for this situation, as well as design some experiments that will allow us to explore the statistical usefulness of spoken n-gram phrases for recognizing mathematics more reliably than text.

The topic of understanding spoken mathematics (but not simultaneously with handwriting) is discussed by Stevens in the Mathtalk project[], and is briefly mentioned in the survey by Kajler and Soiffer [].

Experimentation with Handwriting

Experiments (September, 2003) with a WACOM tablet (Graphire 2 model) on handwriting provide another insight into how easy (or hard) it would be to write mathematics using the built in Microsoft handwriting API. We observe that in particular there is no mode in Microsoft Office tools by which handwritten text is absorbed into the application except as a linear string of words. Thus x^2 which requires the input of characters on two lines is not possible. There is a drawing component which allows a user to create (by handwriting on a drawing pad) the image x^2, and insert it into the text as we have just done here. It might also be possible to write the material into the text in place by writing the mathematics as x squared as we just did.

Here are some examples of how well and how poorly the Microsoft handwriting recognition program works (Sept. 15, 2003). The handwriting tool allows you to revisit a result and ask for possible corrections. In the first three examples, we find it nearly impossible to write "a+b". On the third try the seventh alternative selection among possible corrections is the expected string of characters. By contrast, "5+6", "1+1=2" and Illinois are correct on the first try. Note that the first three letters of Illinois are indistinguishable as symbols from each other as well as from the numeral one or the vertical line in absolute value. Thus recognition of Illinois would challenge isolated symbol programs. Also note that "1/2+2/3" has the correct result as the second alternative, and A/B+C is recognized as two phrases, "A/" is the 4th alternative for the first part, "B+C" is the 6th alternative for the second.

Handwriting	Result	Correct Alternate
a+b	Atb	--
a+ b	At b	--
a+b	Atb	7
5+6	5+6	1

$\sin x$	Sinx	--
$\alpha+\beta$	ants	--
$1+1=2$	1+1=2	1
Illinois	Illinois	1
$1/2+2/3$	112+213	2
$x+y$	4th	--
$X+Y$	458	7
$A/B+C$	Al Btc	4,6
$\frac{a}{b}+C$	Fe,	--

The handwriting API has settings for the recognition of separate characters (as is, for example, necessary for Chinese), thus we are not totally at a loss for potential software re-use. On the other hand, it may be more fruitful to work from a more targeted and simpler technology base, such as FFES or JMathNotes in which the relationship of characters on the screen is more nuanced. In particular, the Microsoft handwriting recognizer assumes a linear stream of character, and thus the best one could hope for out of $\frac{a}{b}$ would be the sequence of strokes a, -, b. And that is not especially probable. Our current plan is to separate stokes ourselves, providing a geometric basis for inserting symbols on the screen (other than at the current insertion point) and then use the handwriting recognizer on the strokes to determine probable symbols for those locations.

From our existing programming experiments, it is clear that the applications for speech recognition and handwriting recognition can be run at the same time; it is also clear that support in the Microsoft framework for two such simultaneous modes has not been a priority.

The Mixture

Mixed or multimodal input has a literature that seems not to include mathematics, but for other domains, see, for example the work of Suhm et.al [21] and Oviatt et.al [22]. A pen device is most often used for a limited repertoire of gestures for pointing or editing. Early efforts in this area have tried to solve the problems of this technology at a rather lower level than we are, as will become evident.

Multimodal Design of Math Input

Consider the handwritten notation $1<2$ which the Microsoft handwriting system recognizes as K2, but with alternatives in this set: {k2, 1<2, 122,12, I<2, 112, ||2, 222}. We would prefer the 3rd choice, namely 1<2 for this form. Consider the spoken "one less than two" which, when I spoke it, was recognized exactly that way. It has alternatives of {one less than to, one than two, when less than two, one less in two, one in two, one < two}. Actually the correct interpretation appears nowhere in that list, at least if we insist on "1<2" and reject "one less than two". It seems clear that it will pay to accept the speech recognizer's spelled-out version. We have a more restrictive grammar that allows the digits "1" and "2" but excludes the words {one, won, Juan, when, …, two, too, to}. Although this cuts down on the range of recognition results, the built-in dictation grammar is quite sophisticated and provides ranked alternatives, not possible with a user-defined grammar.

The basic design looks like this:

A main system processes events on a shared (multiprocessing) queue. The event queue includes data, including the beginning time and duration of the events for each of the following modes:

Handwriting recognition results, the alternative recognition results.

Speech recognition results, the alternative recognition results.

Keyboard events.

Mouse events. mouse events, will, in most configurations, be stylus events that occur outside the handwriting "writing pad" regions. For example, one can select from a menu using the stylus.

Handwriting or speech gestures or commands. The conventional dictation grammar for MS allows the phrase "voice command" to shift to a mode where one can say "File, Print." To return to dictation one would say "Dictation". It may be plausible to design a system that allows "Dictate Mathematics". The conventional handwriting recognizer can provide access to commands based on gestures, where a "scribble" can delete an object. Knowing when and where to allow a scratchout gesture in a natural way requires some careful design. In fact, pointing may be especially important in an editing mode where one might point at a subexpression and say "replace this with 23" or "raise this to the power n".

The main process controls the display and renders the mathematics. Each handwriting event (which may normally be segmented into "stroke neighborhood" and "content" must be examined for correlation with zero or more speech events, past, present and future. Each speech event must similarly be examined for correlation with zero or more handwriting events (It is less likely that a speech event will precede its associated handwriting event. We expect that the speech events will generally follow the corresponding handwriting, by zero to (say) two seconds.)

The processes that put events on the queue are built out of existing technology: currently we are using

1. The Microsoft Tablet PC handwriting recognition software (note: we are not actually using a Tablet PC, but an add-on WACOM tablet). This handwriting application knows nothing about mathematics, and a+b is rendered as, in preference order {`Atb, atb, ate, alb, auth, att., antsy, Atbs,a+b`}
2. The Microsoft Speech SDK 5.1 package.

It is not our intention to teach the handwriting system about our math input: there are no tools for this purpose in the Microsoft package, and the prospects of pushing the prototype demonstrations systems such as JMathNotes further, is daunting. Neither is it our intention to build and train a system that would "fuse" the input data at this early stage where we can adjoin to the "ink" information for + the "audio" information "plus" to yield the unique symbol "+" While it is plausible that one could train something like a neural network to handle the audio and ink information as just "more bits" it seems to us that the separate technologies that have been developed at considerable expense for ink and for speech separately could more easily be used for further development. This is not to say that a combined approach might have better accuracy, but only that building such a system would require major effort in a tangent to most other needs, and we would like to first see if re-use of technology can get us substantially higher accuracy.

The voice-recognition system development kit (SDK 5.1) from Microsoft at first glance seems far more flexible for the programmer, allowing for the development of new grammars and word-lists. Unfortunately, writing one's own grammar in the provided fashion disables the capability we require for the return of alternatives to the recognition. We are seeking a way around this.

The handwriting system development kit is available only for use on the Tablet PC operating system, although the developed code (recognizers) can apparently be run on other (Windows 2000, XP) systems.

Current Status

As of April, 2004, we have a main process control program that displays mathematics entered with a combination of keyboard and mouse events, which also allows speech input. That is, one can point to a location (perhaps a pre-existing expression in a box) on the screen and speak into it. The first alternative of the spoken utterance is inserted in the box.
The main program is written in Common Lisp.

Because of peculiarities in the Speech and/or Lisp development environment, peculiarities that we hope to iron out soon, speech is not yet nicely integrated into this system. Ideally we would like speech to run as a separate process *within* Lisp's multiprocessing paradigm, inserting material as Lisp (or "foreign") data into our event queue. Instead the current implementation for the speech recognition process is written as a half-page of server-side jscript, using Microsoft's Automation facility. It writes the results into a file. This jscripted process is started/stopped from Lisp as necessary; within the Lisp is a process that grabs data when the speech-results file is written.

The lower-level components of the handwriting mechanism provided by Microsoft, namely "Ink" collection and stroke information, can be used nearly directly from Lisp. What is needed is a linkage from Lisp to the handwriting API so that Lisp can ask "What does this ink represent in ASCII" (and what are alternatives to it). The handwriting recognition API development environment is needed for this next step, and requires a Tablet PC, which we have been promised by Microsoft.

Discussion of program design issues remaining, assuming we have solved the display issues, and that voice and handwriting data are enqueued, with alternatives, on an event queue.

a. As items appear in the event queue we cannot process them individually: we must generally wait for additional potentially correlated events, for some period of time. If we wait too long, the system may seem sluggish. We could try decoding the meanings as soon as possible, and post them, subject to correction. If we cannot make sense of the utterance, the user should have some clue as to what is going on. A small window that has speech and handwriting results, incrementally, separately, and combined, might be useful. How might the user might intervene and correct material before it is committed?
b. When we actually find a correlation between voice and handwriting (by statistical estimation of the most-likely common contents of the queue among various alternatives), we presumably have a rank-ordered set of alternatives of which we provide the most likely, but with some backup choices. What do we do with those alternatives? (Maybe make them available with corrections, as well as the speech-only, or handwriting-only possibilities, if that includes other possibilities.)
c. When we post the result(s) to the display window, what exactly do we do with the raw data? (We could display the handwriting or just its recognized version; we can have a hyperlink to the voice.)
d. Given the multimodal version of the data on the display, what should we really do to dump the information to (say) MathML or a computer algebra system? We could reduce it to "merely" text or try to convey the full data. This latter version might be vital if we are (for example) trying to manage an examination and need to look at potentially mis-recognized expressions for partial credit. Thus having the handwritten material might be vital.

Acknowledgments

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Thanks to Lucy Zhang for participating in early discussions. Ms Zhang was supported in part by an undergraduate research grant from the College of Engineering, UC Berkeley. Richard Fateman's visit to ORCCA, University of Waterloo and University of Western Ontario, June, 2003 was useful in refining some of these ideas.
Students participating in this project include Kevin Lin, Michael Chen, Ojan Vafai, Cassandra Guy, Grigoriy Krimer, Steven Stanek, Michael Jurka.

Conversations with Kuansan Wang, Paul Viola, Cory Linton of Microsoft have been helpful. I also thank Richard Zanibbi for comments as well as updates to FFES.

References
(reorder these eventually)

[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/projects/natural-log/

[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 , "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 America. 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
See also T.V. Raman. "An audio view of (La)TeX documents," TUGboat 13, no. 3 Proc. of the 1992 Annual Meeting, 372-379. (1992).

[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," *Proc. Int'l Conf. On Document Analysis and Recognition (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.
Also
H. Okamura, T. Kanahori, M. Suzuki, W. Cong, F. Tamari, "Handwriting Interface for Computer Algebra systems," see reports and software download at
http://infty.math.kyushu-u.ac.jp/index.html

[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, *ICDAR97.*

http://www-sop.inria.fr/cafe/Stephane.Lavirotte/Ofr/root.html also, On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars,
in *ICDAR99*

[20] L. Zhang and R. Fateman, "Survey of User Input Models for Mathematical Recognition: Keyboards, Mice, Tablets, Voice" July, 2003 (draft)
[21] Bernard Suhm, Brad Myers, Alex Waibel, "Multimodal error correction for speech user interfaces," *ACM Transactions on Computer-Human Interaction (TOCHI) (1)* 2001 60-98.

[22] Sharon Oviatt, Phil Cohen, Lizhong Wu, John Vergo, Lisbeth Duncan, Bernhard Suhm, Josh Bers, Thomas Holzman, Terry Winograd, James Landay, Jim Larson, David Ferro, "Designing the User Interface for Multimodal Speech and Pen-Based Gesture Applications: State-of-the-Art Systems and Future Research Directions," HUMAN-COMPUTER INTERACTION, 2000, Volume 15, 263–322. Also

Sharon Oviatt, .Multimodal interfaces. in *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*J. Jacko and A.Sears, eds. Lawrence Erlbaum Assoc., Mahwah, NJ, 2003, chap.14, 286-304.

[23] M. Suzuki, Infty Project, http://infty.math.kyushu-u.ac.jp/index-e.html

[] N. Kajler and N. Soiffer.
"A Survey of User Interfaces for Computer Algebra Systems," *J. Symbolic Computing 25* (2): 127-159 (1998) also see Norbert Kajler (editor). Computer-Human Interaction in Symbolic Computation, Springer-Verlag, Wien, New-York, 1998. ISBN: 3-211-82843-5.

[] R. Stevens, Mathtalk:
http://www.cs.york.ac.uk/maths/robert/mathtalk.html
R. Stevens and A. Edwards, "A Sound Interface to Algebra," also
[] L. A. Chang, Handbook for Spoken Mathematics , Lawrence Livermore Laboratory, The Regents of the University of California, 1983.

[] Mathtype and WebEQ,
http://www.dessci.com/en/products

Richard Zanibbi, Dorothea Blostein, and James R. Cordy. "Recognizing Mathematical Expressions Using Tree Transformation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 11, pp. 1455-1467, November 2002.

Appendix: Some AsTeR examples:

$(a^3+b^3 = (a+b)(a^2-ab+b^2)$ is spoken as "a cubed plus b cubed equals quantity a plus b times quantity a squared minus a b plus b squared. The smaller typefont is used to indicate a voice with lower volume. Notice the use of the word "quantity" to signal a grouping. Also notice that "a b" is the same as "a times b".

(1+sqrt(5))/2 is spoken as "fraction one plus square root 5 divided by 2."

$\int_1^\infty e^{x^2-x} {}^{-1}\, dx$ is spoken as "Integral from x equals one to infinity of e raised to x squared minus x minus 1 d x." In this case the smaller font is read in a higher (squeaky) voice.

6/4/2004 1:55:02 PM, Richard Fateman