Math Boxes: A Pen-Based User Interface for Writing Difficult Mathematical Expressions

Eugene M. Taranta II
University of Central Florida
Orlando, FL 32816 USA
etaranta@gmail.com

Joseph J. LaViola Jr.
University of Central Florida
Orlando, FL 32816 USA
jjl@eecs.ucf.edu

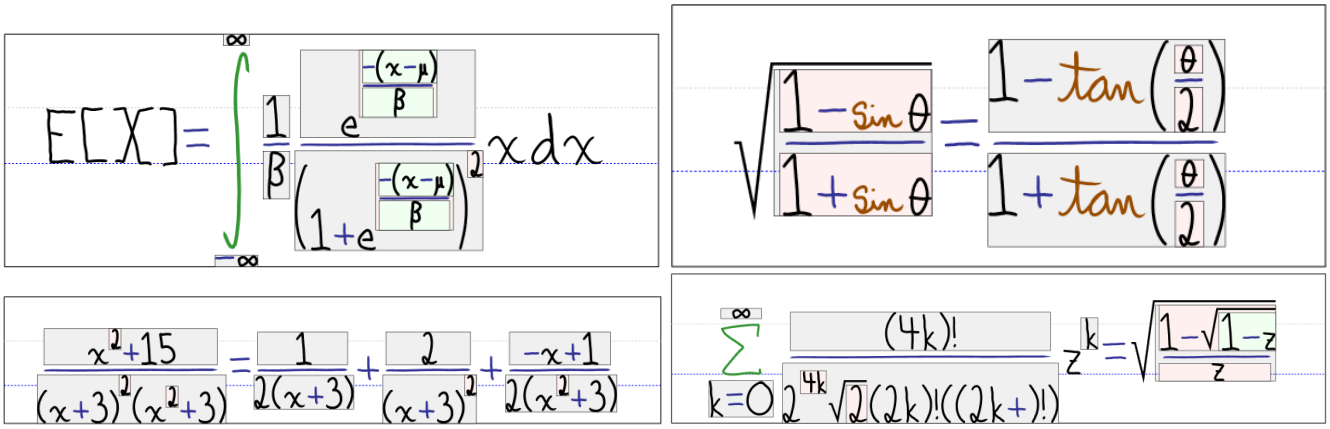


Figure 1: Four examples equations utilizing math boxes. Subexpressions, according to their spatial relationship with neighboring terms, are entirely contained in a hierarchy of boxes. A math box and its parent boxes are dynamically resized when the user hovers over a box's boundary so that the associated subexpression can be easily extended.

ABSTRACT

We present *math boxes*, a novel pen-based user interface for simplifying the task of hand writing difficult mathematical expressions. Visible bounding boxes around certain subexpressions are automatically generated as the system detects specific relationships including superscripts, subscripts, and fractions. Subexpressions contained in a box can then be extended by adding new terms directly into its given bounds. Upon accepting new characters, box boundaries are dynamically resized and neighboring terms are translated to make room for the larger box. Feedback on structural recognition is given via the boxes themselves. We also provide feedback on character recognition by morphing the user's individual characters into a cleaner version stored in our ink database.

To evaluate the usefulness of our proposed method, we conducted a user study in which participants write a variety of equations ranging in complexity from a simple polynomial to the more difficult expected value of the logistic distribution. The math boxes interface is compared against the commonly used *offset typeset (small)* method, where recognized expressions are typeset in a system font near the user's unmodified ink. In our initial study, we find that the fluidness of the offset method is preferred for simple expressions but as difficulty increases, our math boxes method is overwhelmingly preferred.

IUI 2015, March 29–April 1, 2015, Atlanta, GA, USA.

http://dx.doi.org/10.1145/2678025.2701400

Author Keywords

Sketch-based User Interfaces; Sketch Recognition; Mathematical Sketching; Handwritten Mathematics

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces - Graphical user interfaces.

General Terms

Human Factors; Design; Measurement.

Introduction

Recognition of handwritten mathematics has a long history [2] and is a critical component of numerous pen-based software interfaces for physics [7, 9], geometric theorem proving [4], Boolean logic visualization and manipulation [5], and algebraic intelligent tutoring systems (ITS) [1] among others. Users of these systems demand high accuracy. For instance, Anthony et al. [1] estimate that 91—97% recognition accuracy is required for acceptance in a mathematics oriented ITS system. However, based on results from the fourth international Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) [11], expression level

recognition rates are still well below this threshold. Complementary to mathematical expression parsing and recognition, user interface (UI) design can play a crucial role in usability and can potentially assist individuals in writing difficult expressions as well as help simplify error correction. That is, a well designed UI can go a long way towards improving a user's experience and acceptance of such systems.

To this end, our work is motivated by two observations. First, without in-depth technical knowledge of the algorithms and architecture employed by the mathematical parser, it can be difficult to determine the source of a parsing error (or set of errors) and consequently, equally difficult to correct the underlying issue. Second, we observe that even when an expression is available to a user beforehand, he or she will not ration sufficient space in advance to write out the full expression. For example, parenthesis are often too small for large inner expressions; fractions bars are too short for long numerators or denominators; and superscripts are initiated too low when they contain fractions. Worse still is that when these issues do occur, moving symbols around into their desired positions or rewriting subexpressions may not be enough to ameliorate any or all errors. With *math boxes*, we aim to address both of these issues.

Math boxes relies on simple structural parsing only to be able to detect superscripts, subscripts, and fraction bars. Once a particular relationship has been determined to exist between two symbols, one in which they do not share the same baseline of math, then boxes are created to contain symbols as is appropriate for the specific relationship; see Figure 1 for a few examples. Math boxes are visible to the user and can be easily extended. Hovering over a box with a stylus or writing into a box causes its boundaries to grow, and neighboring ink that overlaps with the box is translated aside, to make room for the expanded box and possibly new symbols. This eliminates the need for complex structural parsing and helps to ensure that users will always have enough room to continue writing long expressions. As a consequence of this approach, error correction is simplified because an expression's structure is tied directly to the math boxes interface, which therefore does not require a guess as to why the underlying recognizer went astray when an error occurs, or trial and error correction.

Based on these ideas, a prototype system utilizing math boxes was developed and subsequently evaluated and compared against the currently best known and highly rated user interface design *offset typeset (small)* [8, 22]. After asking participants to write a range of equations varying in complexity, we find that participants prefer using math boxes for difficult equations and further, accuracy and time to completion improves as expressions become more complex, as compared to the offset typeset (small) interface.

RELATED WORK

Recognition of handwritten mathematics has been an active area of research for many years, with most effort focusing on techniques for parsing characters and structure [16, 19], though less attention has been given to UI techniques for providing recognition feedback and editing features. Zanibbi et al [20] introduced the concept of style preserving morphs to provide instant feedback on the system's interpretation of an expression's structure. Individual characters are resized and translated into their correct structural position, however, this technique does not indicate what symbol was actually recognized nor does it help to prevent errors. Further, by moving characters into their correct structural position, users may not have enough space to extend a subexpression.

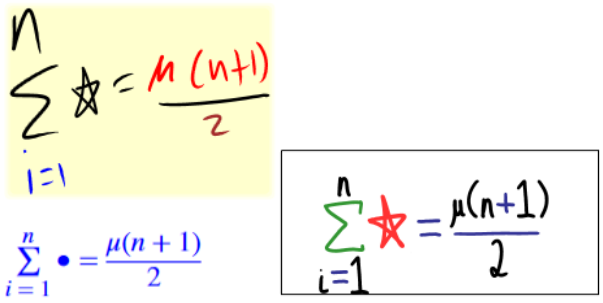
Smithies et al.'s Freehand Formula Entry System (FFES) [14, 15] is a pen-based interface for editing equations. The system supports four modes of operation: draw, modify stroke groups, modify characters, and select and move (for equation level error editing). FFES, therefore, does not provide a seamless integration between recognition and editing. Labahn et al.'s MathBrush [6] is designed specifically to work with a computer based algebra system. One interesting feature of their UI is that users have the ability to circle subexpressions and work with them separately. Although this was designed for algebraic manipulation, it is a potentially useful UI technique that is slightly related to math boxes in that boxes help to isolate subexpressions.

Unlike most previous work that focuses on the construction of equations, Zeleznik et al.[22] focused specifically on UI design for feedback of recognition results. To the best of our knowledge, this work represents the state of the art. In their work they describe several variations of typesetting. These include replacing recognized characters with system font characters or prerecorded ink characters, and typesetting the recognized equation underneath the user's ink in a system font of varying size. They also experiment with ink color as well as techniques for dealing with allographs. All of these variations were thoroughly evaluated by LaViola et al. [8] who found that *offset typeset (small)* was the preferred choice of recognition feedback, which is described in the next section. Zeleznik et al.[21] introduced Hands-On Math, a pen and touch bimanual interface that makes computer algebra systems accessible to users through virtual pages. They developed a rich set of interaction techniques to manipulate algebraic expressions and manage virtual pages. However, the error correction and space issues discussed previously that are handled by math boxes are not addressed in their work.

Non pen-based interfaces also exist that supports structural mechanisms akin to math boxes. For instance, BREDIMA [12] is a web-based tool for editing mathematical expressions where each math object contains its own rectangular region, though actual structural changes still requires various WIMP interactions. We are unaware of any other pen-based UI technique that structures mathematical expressions utilizing interactive and dynamic boxes as we describe.

OFFSET TYPESET (SMALL) USER INTERFACE

Zeleznik et al.'s offset typeset (small) user interface [22], or just offset hereon, is a well studied interface and is the preferred feedback interface for recognition of handwritten mathematics [8]. Their design provides recognition feedback in a number of ways. Most notably, results from the math recognizer are displayed in a typeset system font underneath the user's ink. In this way a writer is not distracted by recognition feedback (such as by character morphing) and he or she can



(a) Offset UI **(b)** Boxes UI

Figure 3: Side by side comparison of various aspects of the the offset (a) and math boxes (b) user interfaces. Specifically, differences in how unrecognized characters are handled, how ink is colored, how each system typesets mathematics, and how much space is required is shown.

verify recognition correctness as is convenient. Further, the user's ink is colored according to the recognized structure of the expression. This allows for quick validation of alignment and various parsing results. An example of the offset interface is shown in Figure 3(a).

To edit an expression, offset provides two techniques. First, scribble-erase allows a user to scribble over previously penned ink, thus erasing those strokes that overlap with the convex hull of the scribble stroke. Second, ink is translated by using the lasso-drag gesture. Targeted strokes are circled and subsequently dragged into the new position. Note that the system continues to parse ink as it is being dragged around, allowing for instant feedback so that the user can stop dragging when he or she is satisfied with the result.

A major disadvantage of the offset interface is that user feedback does not provide enough information as to what errors have occurred when the typeset equation does not match the user's expectation. Therefore users have to guess as to which strokes or arrangements may have caused their error. This violates Wais et al.'s [18] guidance that recognition errors must be predictable and/or understandable. Wais et al. also report that users generally prefer to correct errors when they are done drawing, which may further exacerbate the problem. In mathematical sketch recognition, delayed error correction can potentially make the task of understanding and correcting an underlying error more difficult.

MATH BOXES USER INTERFACE

Math boxes are dynamic containment units for subexpression symbols that share a common spatial relationship. For instance, consider the expression:

$$x^{2+ab} + \frac{y^3}{z} + w^5. \quad (1)$$

In this equation, the symbols x, $+$, w, and the fraction bar share a common baseline of math and consequently, a common math box. Symbols of the subexpression $2 + ab$ are offset from the previous math box, being a superscript of x, and therefore also share their own math box. Note, however, that even though 5 is offset in the same manner, it is a superscript of a different symbol and so will reside in its own math box. For similar reasons the numerator, denominator and 3 superscript all reside within their own math boxes. In total, the expression of Equation 1 is comprised of six unique math boxes.

For reference throughout the remainder of this section, the flowchart diagrams in Figure 4 show how changes in the stylus hover state and how new strokes are handled. Most notably, math boxes are created on-the-fly as new symbols are added to an expression. Since the boxes are visible to the user, they also convey useful information about the recognized spatial relationship between the various symbols. For example, a user can be certain that a new symbol is recognized as a superscript because the new symbol is instantly wrapped inside of a new math box.

Recall that one goal of this work is to eliminate space planning issues for when a user does not leave his or herself enough room to write a large expression. To accomplish this goal, each math box dynamically resizes itself as symbols are added into a math box's boundary. However, as boxes are resized, they may expand into and overlap with other already existing symbols. This clutter and confusion is avoided by translating neighboring ink out of the way, to make room for an expanded math box as is depicted in Figure 5.

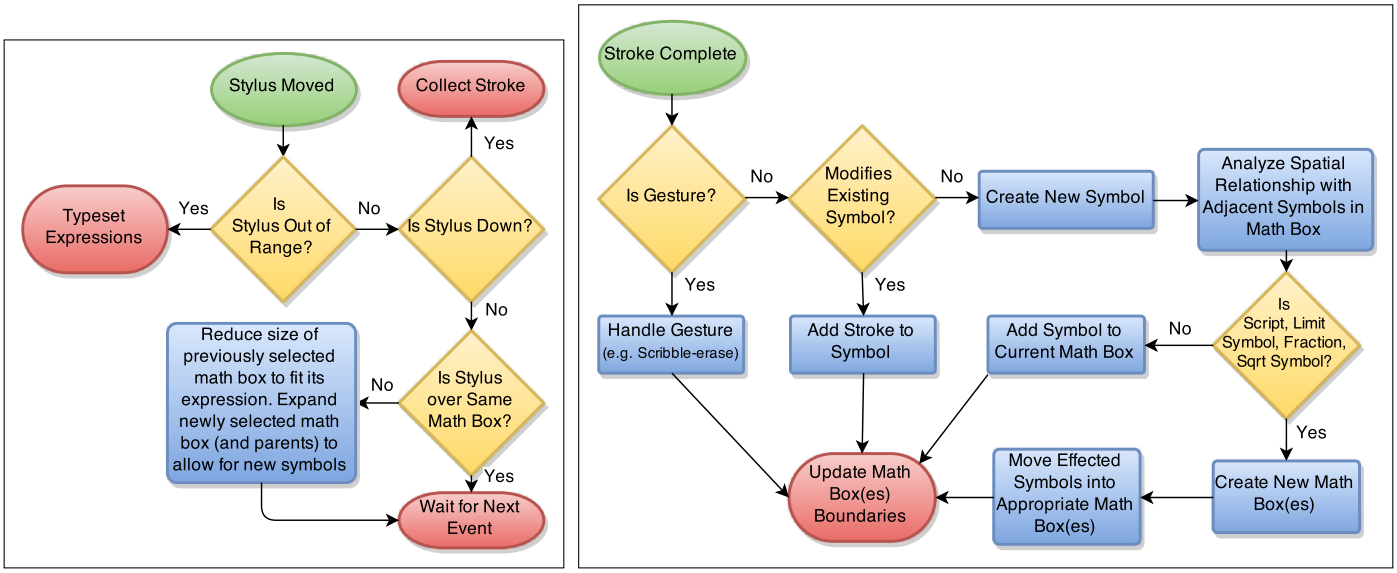
Superscripts and Subscripts

New strokes that are determined to exist in the superscript or subscript space of the left adjacent symbol will cause an associated math box to be created. The subexpression of either the superscript or base can then be extended by writing into the appropriate math box. If a user pens what he or she expects to be a subscript or superscript and a math box does not appear, then the user instantly knows the stroke was not parsed correctly. If this goes uncorrected, then the symbol will be brought down to the baseline at a later point in time (when the expression is typeset).

Only certain symbols can contain superscripts or subscripts. Alphabetic characters can have both whereas numbers can only have superscripts. Left brackets (rounded, curly, or squared) cannot have either and closing brackets can have only superscripts. These restrictions are not only reasonable, supporting most mathematical notation requirements, but also help to eliminate various parsing errors.

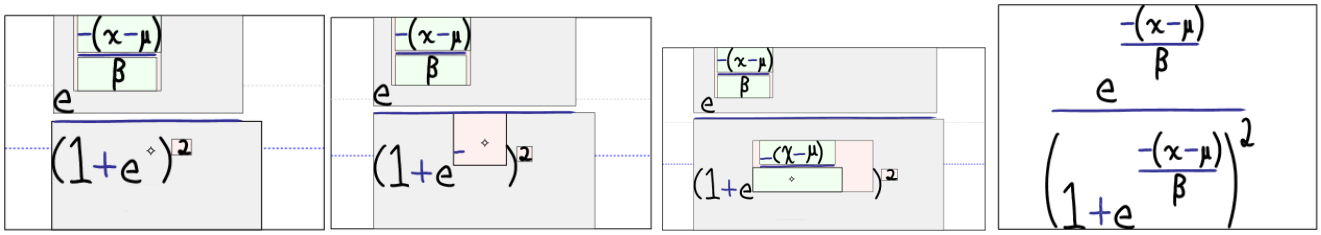
Fraction Bars

Fractions generate two math boxes, one for the numerator and another for the denominator. If a horizontal line is drawn and there are no symbols above or below the line, then the stroke is assumed be a negation or minus sign operator. On the other hand, when a character is written above or below the line, then both math boxes are created and the new character is added to the appropriate math box. Another way to work with fractions is to first write one or more symbols of the numerator or denominator and then draw the fraction bar. Any symbols that overlap with the projection of the fraction bar onto the horizontal axis are automatically added into the appropriate



(a) Hovering stylus. **(b)** New stroke collected.

Figure 4: Flowchart for stylus events when (a) the stylus is hovering and when (b) a new stroke is collected. Details of the system are discussed throughout math boxes user interface section.



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

Figure 5: An example of writing a complex expression. A typical scenario (a) occurs when the user fails to reserve enough space for the denominator $(1 + \exp(-(x - \mu)/\beta))^2$. In (b), once the minus sign is recognized in e's superscript space, a superscript box is generated, providing enough additional space so that the subexpression can be further extended if desired. After the fraction bar is drawn (c), terms above the line are moved into a numerator box. Similarly, a denominator box is supplied as surrounding ink is simultaneously pushed aside. Finally in (d), by moving the stylus out of range, our system typesets the expression.

math box. Further, the fraction bar automatically resizes as the numerator or denominator grows so that users can be certain of and confident in the relationship between the math boxes.

Limits

Limit boxes are automatically created for sigma and integral symbols. These symbols have to be drawn first, before any limits are written. If the limits are written first, then we assume these symbols were intended to be part of the math box's baseline of math. However, this is something we intend to address in future versions of the prototype system.

Symbol Morphing

Many users in fact do not prefer to see their handwriting changed [8], but without typesetting the equation separately (and thus wasting space in user interface), we are unaware of a better method to provide feedback concerning character recognition correctness. Therefore we choose to morph the user's symbols into cleaner versions of the recognized characters stored in our database of prerecorded examples. This differs from style preserving morphs [20] since recognized characters are not only rescaled and translated, but they are also reshaped. To avoid annoyances with rapidly replacing the original character, the morph is instead animated. Dynamic timing warping [17] is utilized to determine which points from the original ink best aligns with the replacement ink characters points, to make the transition as smooth as possible.

Typesetting

When the stylus is taken out of range of the tablet PC, then any modified expression is typeset. All symbols are properly aligned according to their 2D spatial relationships and spacing between symbols is corrected. Because users do not write uniformly with proper variation between offsets (e.g., from the baseline to the superscript space), most symbols also have to be resized. To minimize the amount of change to each character,

we utilize the golden section search [13] such that the change in character height is the error to minimize. Consequently, the baseline capital letter height is selected to be the value that minimizes the height change of all symbols. We tried similar techniques for determining the baseline and left offset but found through iterative design that utilizing the initial character's original position for the baseline and left offset worked best. Otherwise an expression's change in position could be too distracting, even if this minimizes the total overall movement of each symbol.

As part of typesetting an expression, parentheses and integral symbols are resized to match the subexpression associated with the operator or group. In both cases however, resizing only occurs when the expression is typeset, not as the user is inking new symbols. As already mentioned above though, this contrasts with fraction bars, which are dynamically resized as new math is added into either math box. Finally, parentheses resizing also only occurs if the group set is complete (i.e., a subexpression with an open left parenthesis will be resized if their is a matching closing right parenthesis).

Typesetting an expression removes excess space between symbols created by the math boxes, but this is not a problem however. If the user thereafter wishes to extend a subexpression, hovering over the associated math box will cause the box to expand again, so that the subexpression can be further modified.

Editing

The same scribble-erase gesture that is supported by the offset interface is also supported by math boxes. However, the lasso-drag gesture is not directly applicable. As an alternative, if a user needs to make room for additional symbols in-between two preexisting characters, he or she can hover-drag a character horizontally to create the necessary space. That is, a user single-taps a symbol he or she wishes to move. The convex hull of the symbol is highlighted yellow and then the symbol is dragged horizontally while hovering over the display. A final single tap releases the symbol into its new position. At the same time, terms left or right of the symbol (depending on the direction of displacement) are also translated and the associated math box expands to accommodate the new configuration. Math boxes also provide a mechanism for easily expansion. If a user feels constrained and requires additional space, he or she can simply tap in an empty region of the math box to inflate the box's boundaries.

EVALUATION

StarPad[1] (stylized starPad) is an open source SDK for the recognition of handwritten mathematics developed for Microsoft's .NET framework and is based on [10, 22, 23]. The SDK comes with an example application that implements the offset user interface described in [8, 22], therefore we utilized this system in our evaluation. Further, to ensure a fair comparison between user interfaces, we developed a math boxes implementation around the same underlying character recognizer.

[1] `http://pen.cs.brown.edu/starpad.html`

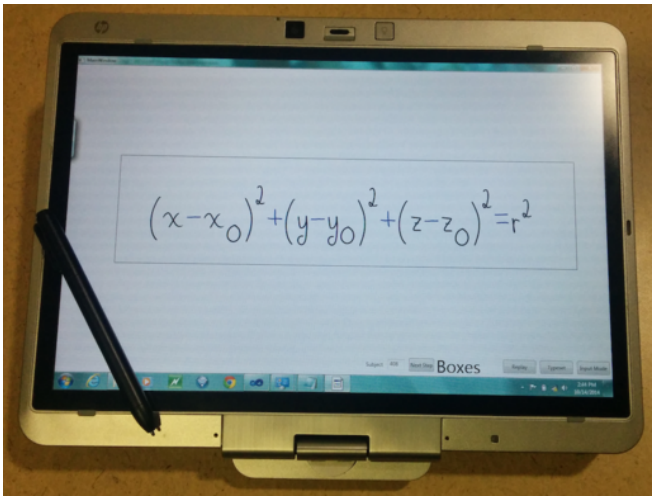


Figure 6: Apparatus utilized for user study with Eq 4 in Table 1 shown.

We conducted a user study to evaluate the effectiveness of math boxes. From the University of Central Florida, we recruited 12 participants, ranging in age from 18—53 (mean=24.67). Nine participants were male and three were female, two left handed and ten right. The participants' backgrounds included communication sciences and disorders (1), health information management (1), computer science (7), electrical engineering (1), statistics (1), and physics (1). Six participants had some prior experience with handwriting recognition systems (mostly for note taking on tablet devices). Each participant took between 90—120 minutes to complete the study and were compensated $20.

The evaluation was conducted on a Hewlett Packard HP EliteBook 2760p Tablet PC with an Intel® Core™ i5-2410M CPU (2.3GHz) having four gigabytes of memory, running Microsoft Windows 7 Professional. The touch interface was disabled so that participants would write naturally without having to worry about interfering with the interface. The tablet PC was rested upon a table and pitched at an adjustable incline so that participants could sit and work comfortably.

Experimental Design and Procedure

We conducted a within-subjects experiment to evaluate user performance and perception of the math boxes interface compared to the offset interface. The independent variable was the interface used to write a set of equations, and the dependent variables were the time to completions, accuracies, and post-questionnaires. Participants first filled out a short pre-questionnaire in order to collect demographic information and prior experience with handwritten mathematics on a stylus or touch device. Subsequently, each participant was then introduced to both user interfaces via two practice problems per method. In this training phase, complex expressions are constructed incrementally through a number of stages that let users become familiar with each system and their editing features. For the offset method, this included explanations on and practice with the scribble-erase and lasso gestures. For the

Eq 1 $(a+b)^2 = a^2 + 2ab + b^2$

Eq 2 $\sum_{i=1}^{n} = \frac{n(n+1)}{2}$

Eq 3 $f(x) = ax^3 + bx^2 + cx^1 + dx^0$

Eq 4 $(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2$

Eq 5 $\int \frac{e^{ax}}{b+ce^a x}\mathrm{d}x = \frac{1}{ac}\log(b+ce^{ax})$

Eq 6 $\sqrt{\frac{1-\sin\theta}{1+\sin\theta}} = \frac{1-\tan\left(\frac{\theta}{2}\right)}{1+\tan\left(\frac{\theta}{2}\right)}$

Eq 7 $\frac{x^2+15}{(x+3)^2\,(x^2+3)} = \frac{1}{2(x+3)} + \frac{2}{(x+3)^2} + \frac{-x+1}{2\,(x^2+3)}$

Eq 8 $\int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-\mu)^2}{2\sigma^2}}\mathrm{d}x = 1$

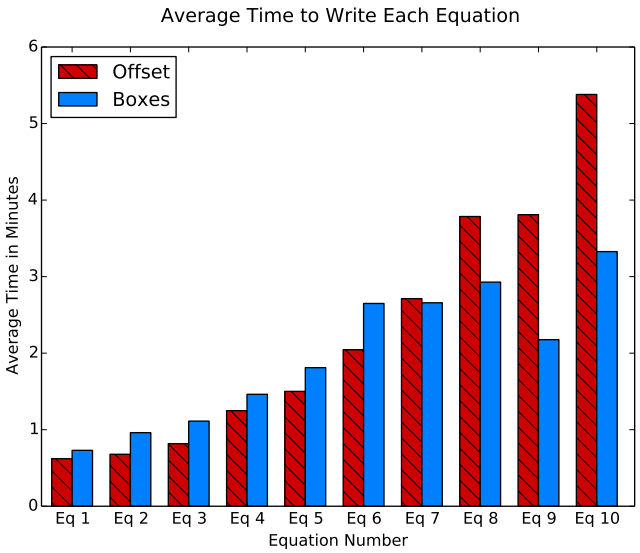
Eq 9 $\sum_{x=0}^{\infty} \frac{(4x)!}{2^{4x}\sqrt{2}(2x)!((2x+1)!)} z^x = \sqrt{\frac{1-\sqrt{1-z}}{z}}$

Eq 10 $E[X] = \int_{-\infty}^{\infty} \frac{1}{\beta} \frac{e^{\frac{-(x-\mu)}{\beta}}}{\left(1+e^{\frac{-(x-\mu)}{\beta}}\right)^2} x\mathrm{d}x$

Table 1: Equations used in the evaluation, ordered from shortest to longest in average time taken to write the equation using the offset method. Note that the equations are labeled to match Figure 7 and Figure 9.

math boxes method, scribble-erase was similarly demonstrated and practiced, but also the hover-drag gesture was introduced. In addition to allowing participants to become familiar with the interface, we also provided guidance on best practices to ensure characters are correctly recognized and how to fix commonly occurring errors. Training lasted between twenty minutes to one half hour.

After having completed training and once the participant was ready to continue, he or she was asked to write ten unique equations twice, once for each method. These equations are shown in Table 1 and vary in complexity but are roughly ordered from easiest to hardest in the evaluation. To ensure that the participant's focus was not drawn away from the screen, the equation under consideration was typeset in a system font and displayed in the upper left corner of the display; the participant was free to write anywhere else on the display. The order of which method was used first for each equation was constant throughout the training and evaluation, however, this order alternated between participants so that half started with offset



Eqs 1—5	t_{11}-values	1.956	2.085	2.920	1.641	1.343
	p-values	0.076	0.061	<0.05	0.129	0.206
Eqs 6—10	t_{11}-values	1.761	-0.184	-1.659	-2.112	-3.655
	p-values	0.106	0.858	0.125	0.058	<0.05

Figure 7: Average time in minutes taken to write each equation using both methods and the T-test results for each equation. The equations are given in Table 1 and are roughly in order of difficultly.

and half started with math boxes. Throughout the evaluation and only on occasion, if the user struggled to correct an error (usually while using the offset method), we would provide suggestions on how to correct the error. Further, because the tablet PC exhibited sluggishness in certain situations (see the discussion section), we would occasionally have to remind the participant to write slower and wait for the dynamic resizing and rendering of the boxes to complete before penning new characters. Once the participant completed writing an equation using the second method, he or she was then asked to provide feedback on a 7-point Likert scale concerning the difficulty of each method.

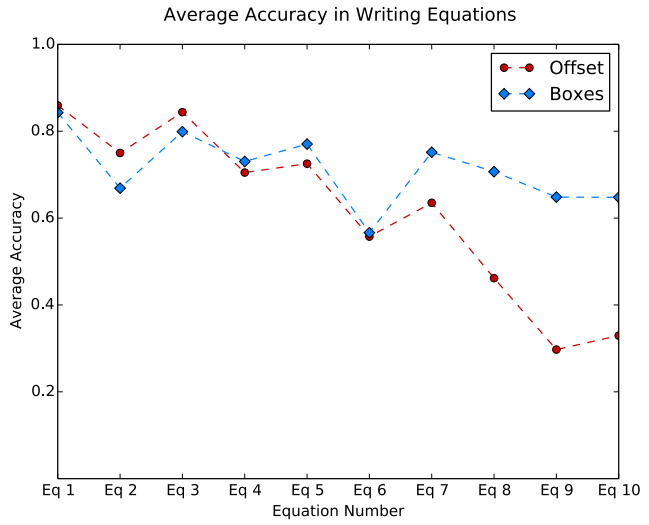
Data Collected

During the evaluation, the time to completion (TTC) for each equation was recorded. We also recorded each modification so that the entire sequence of events could be replayed if necessary. The number of modifications is also used to calculate the error rate as discussed in the results section. For each equation we also collected immediate ease of use feedback for each interface. Finally, in a post-questionnaire we collected information on general/overall impressions.

Results

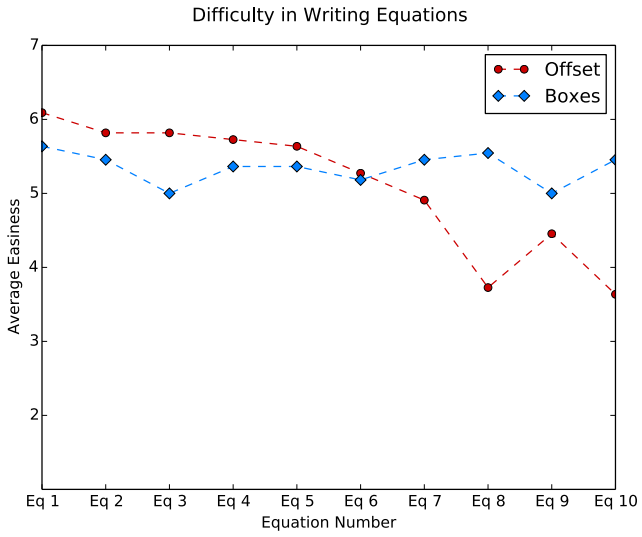
Quantitative Results

For each equation we recorded time to completion (TTC) and total modifications required to complete the equation. Figure 7 gives the average TTC results in increasing order by offset TTC as well as the T-test results for each equations. For simple and moderately simple equations, the TTC results



Eqs 1—5	t_{11}-values	1.412	0.789	-0.471	-0.493	-0.136
	p-values	0.816	0.188	0.448	0.648	0.632
Eqs 6—10	t_{11}-values	-0.136	-2.007	-3.007	-2.720	-5.175
	p-values	0.895	0.073	<0.05	<0.05	<0.05

Figure 8: Average accuracy for each equation and the T-test results for each equation. Accuracy is calculated as the minimum number of required strokes to complete an equation over the total number of modifications.



Eqs 1—5	z-values	10.500	7.000	10.000	25.000	18.000
	p-values	0.281	0.206	0.130	0.449	0.584
Eqs 6—10	z-values	9.000	9.000	2.000	10.500	0.000
	p-values	0.739	0.196	<0.05	0.286	<0.05

Figure 9: Perceived difficulty of writing each equation for both methods where 1 is very hard and 7 is very easy, and the Wilcoxon signed-rank test results. Notice that the offset method trends downward as the equations become more complex whereas boxes remains consistent.

are similar even though math boxes is slightly slower (reasons for this are outlined in the discussion). In general, for the more complex set of equations, there is a noticeable difference in TTC rates where math boxes outperforms offset. Even though we only observed statistical significance for equation 10 ($t_1 1 = -3.655, p < 0.05$), we believe this result is representative of the true distribution. There were also two instances where a participant gave up on an equation, in both cases the offset method was under consideration and in both cases the same equation was completed using the math boxes interface.

There are a number of approaches to calculating accuracy [3]. Since we are concerned with reducing difficulty in writing complex expressions, we utilize an accuracy measure that relates to the total amount of work a participant needs to do to write an equation and therefore, we rely on modification counts. A modification is any action that results in an invocation of the parser, such as when a user pens new characters, erases previous ink, or repositions symbols via the lasso-drag gesture. To calculate accuracy we first determined the minimum number of strokes required to complete an equation, which is equivalently the minimum number of modifications needed to complete the equation. The minimum is then divided by the total number of modifications observed so that accuracy is in terms of effort. Using this metric, accuracy results from the evaluation are shown in Figure 8. For simpler equations, where structural parsing errors are less likely, the accuracy of both methods are approximately equivalent. However, as the equations become more difficult, accuracy diverges such that math boxes is significantly more accurate.

Also notice that offset continues on a downward trend as the complexity increases which contrasts with math boxes that is moderately consistent across all equations. One exception, however, is Eq 6 which was especially subject to the issues covered in the discussion section.

Qualitative Results

In addition to TTC and accuracy, we asked each participant to rate how easy each user interface was to use for each equation, where 1 is very hard and 7 is very easy. These results are shown in Figure 9. The outcome here matches the quantitative data in that for simpler and moderately similar equations, there is little difference. Offset has a slight advantage with higher scores, but the difference is not statistically significant. This is consistent with our expectations as we would not anticipate math boxes to be significantly harder to use for these equation types. For complex equations, however, using the Wilcoxon signed-rank test, math boxes is reported to have an advantage over offset with two of the complex equations—they have statistical significance ($Z = 2.0, p < .05$) and ($Z = 0.0, p < 0.05$ for Eq 8 and Eq 10 respectively). As before, offset's ease of use trends downward from approximately 6 to below 4 as complexity increases, which contrasts with math boxes that remains fairly level with some fluctuation around 5.5.

Participants were asked via a post-questionnaire to rate their overall impressions of both systems, for specific features and in terms of ease of use. The results of this questionnaire are shown in Table 2. All questions have a value range of 1—7, where 7 is generally more favorable. The offset method

Overall (1=hard, 7=easy)	Offset	Boxes	Z	p
Ease of use	4.417	5.5	5.5	<0.05
Ease in identifying errors	5.50	5.667	15.5	0.723
Ease in correcting errors	3.583	5.5	0.0	<0.05

Preference (1=offset, 7=boxes)	Offset	Boxes	Std. dev.
For simple equations	1.545	–	0.988
For complex equations	–	6.364	1.494

Typesetting (1=disliked, 7=liked)	Offset	Boxes	Z	p
Typesetting System	5.5	5.167	4.0	0.336

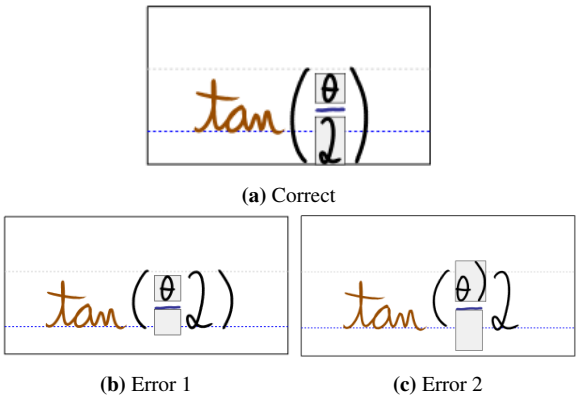
Math boxes features (1=disagree, 7=agree)	Boxes	Std. dev.
Liked character morphing	5.08	1.61
Boxes were comfortable	5.33	1.49
Boxes were not distracting	5.25	1.64
Boxes were helpful	6.000	0.912

Table 2: Results of post-questionnaire. Each question has a value range of 1–7 for which the meaning (given in the table) varies per question. The mean is shown for question and where applicable, the Wilcoxon signed-rank test (Z), otherwise, the standard deviation.

received a moderate score of 4.417 in overall ease of use, whereas math boxes achieved a more enthusiastic response, scoring 5.50 with statistical significance between interfaces using the Wilcoxon signed-rank test ($Z = 5.5, p < 0.05$). This is despite similar ease of use ratings, completion times, and accuracy for most equations. While ease in identifying errors were positive for both systems (5.5 and 5.667 for offset and math boxes respectively with $Z = 15.5, p = 0.723$), there was a much greater contrast for ease in correcting errors. With high significance ($Z = 0.0, p < .05$), offset received a moderately negative score at 3.583 and math boxes achieved 5.50. This may directly relate to why math boxes is considered easier to use overall. For equations that are considered simple as defined by the participant's intuition, offset is preferred over math boxes unanimously and the converse is similarly true. Math boxes, as equations become complex, is strongly preferred by all participants.

With respect to each interface's typesetting approach, offset averaging 5.5 and math boxes closely trailing at 5.167 ($Z = 4.0, p = 0.336$), there seems only small (without significance) preference towards offset. This difference, with offset having a slightly higher average, is consistent with LaViola et al.'s findings [8], where users report offset-variant interfaces to be less distracting and less frustrating than ink modifying interfaces.

A subset of post-questionnaire questions focused on specific features of the math boxes interface. Participants gave a moderately positive response to symbol morphing, 5.08 with $\sigma = 1.605$. Note that one issue identified in symbol morphing was that participants would rather see characters morphed into their own handwriting. Overall, participants found that working in and with boxes was very helpful (scoring 6, $\sigma = 0.91$) and that, despite their dynamic resizing, the boxes were not too distracting (5.25, $\sigma = 1.64$). Though the features and interface provided by math boxes gained positive reviews, there were still a number of issues that caused confusion and frustration, that kept math boxes from earning higher scores. The discussion section goes into detail on these issues.



(a) Correct

(b) Error 1 **(c)** Error 2

Figure 11: Example errors that occurred using the math boxes method. The correct form is shown in (a). In (b) and (c) the 2 is drawn faster than the fraction bar can be registered, before the denominator box is rendered. This results in the 2 being accepted as part of the top level white box, as a character inserted after "$\tan(\frac{\theta}{}$". In (c), the numerator box containing θ is registered but not yet rendered as the user begins writing the closing parenthesis. Because the stylus lands in the numerator box's bounds, the closing parenthesis is accepted into the numerator box despite not being visible.

DISCUSSION

Math boxes was designed to address space constraint issues that arise naturally in writing long expressions and to simplify error correction, so that having intimate knowledge of the the underlying parser is not necessary to fix even trivial errors. The unanimous conclusion based on participant feedback is that math boxes accomplishes its goal. This is also reflected in the time to completion results, in the accuracy results, and in the per equation ease of use surveys for complex equations. In discussions following the evaluation, many participants stated that the structure provided by math boxes was helpful, especially when the expression contains multiple levels of subexpressions. Math boxes is not a perfect system however.

The offset method was said to be considerably more fluid and was preferred by all participants for writing simple equations, a result that is reflected in both the quantitative and psychometric data. Many also felt that a hybrid system might offer a better alternative, so that one can write fast in normal circumstances but be able to utilize a yet unspecified gestural type of mechanism to create math boxes when needed.

Observations, Feedback, and Suggestions

The system that we used in our evaluation may have been the cause of a number of errors and difficulties that occurred during the evaluation of math boxes. Some of these issues are depicted in Figure 11. Namely, the primary issues were due to sluggishness in the system. With the collection and processing of each new stroke, the associated boxes have to be dynamically resized. In many cases, to make room for larger boxes, surrounding ink also needs to be translated (so that the

boxes do not overlap with other characters). When new boxes have to be created, in some of the worst case scenarios, the entire process of handling a new stroke can take as long as 600–1000ms to complete. Because users are not accustomed to having to wait for the system to catch up to write new characters, participants would often get ahead of the process and wind up writing into blank areas that would normally have a box (such as in a denominator space), but before the box actually exists. In another common scenario and for similar reasons, a box expansion would be in progress with the final boundaries already defined. However, the box would not yet be fully rendered and so the user, believing he or she was in the outer box, would add a character to the canvas that was actually accumulated into the inner box not yet visible. We believe that if such updates had occurred in real-time, then the overall times for the math boxes method would be faster and less errors would have occurred. We found through instrumentation that the majority of the latency came from interacting with ink points (.NET WPF components). In other words, the time it takes to calculate the new position of each ink point when typesetting, morphing, expanding, and so forth is negligible.

Another issue that occasionally caused frustration occurred when a user accidentally drew his or her fraction bar imprecisely, so that characters left or right of the targeted subexpression were unintentionally accumulated into the numerator or denominator box. Presently, the only way to correct this error is to scribble-erase the character and rewrite it into the correct position. To help with this issue and those previously discussed, and based on discussions with the participants, having an easy to use gesture to simply move symbols between adjacent boxes would have likely improved the overall user experience.

The math boxes system was designed so that characters could be modified by writing over previous characters. Therefore when a new stroke overlaps with previously written ink, the new stroke is accumulated into the character space and the previous character is either modified (if the additional stroke results in a valid character) or completely overwritten. In this regard, our system is too rigid as this sometimes results in accidents where users unintentionally overwrite neighboring characters.

A final issue that caused frustration and sometimes errors was due to the automatic and frequent typesetting. Users would periodically raise the stylus out of range of the tablet, causing the system to typeset their equation, and then bring the stylus back down to continue writing. Because symbols and their associated boxes had been moved around, the participant would land the stylus in the wrong position. Usually this was quickly noticed and corrected with a scribble-erase. However, a separate gesture or delayed typesetting timer mechanism could easily resolve this issue and, again, further improve the user experience.

Despite these issues and limitations, participants still preferred working with math boxes as equations became more complex. This response is reflected not only in the post-questionnaire results, but also in the accuracy and TTC results. Participants stated that math boxes's dynamic resizing and ink transformations greatly simplified situations involving multiple levels of subexpressions such as when a term in the denominator of a fraction possesses a superscript.

Future Work

In addition to addressing those issues aforementioned discovered during the user study, we wish to explore techniques that increase the acceptance of morphing recognized symbols into an alternate form for the purpose of recognition feedback. We are also interested in methods of typesetting parts of the equation that are not being modified in realtime without causing frustration or distraction, which we believe is an important part of space management, readability, and thus of the user interface. Last, the current design requires the interface to track an input device while it is hovering over the display. Many systems do have this support such as most mobile devices. Therefore, a math boxes interface that does not require hovering is another important avenue of research.

CONCLUSION

We have presented math boxes, a UI design for handwritten mathematics to make identifying and correcting errors easier and to assist users with writing difficult equations. In a user evaluation, we discovered that we did indeed make significant progress towards our goals but we also discovered several ways to improve the interface. While all users preferred to use our math boxes interface for complex equations, they still preferred the fluidness of the traditional offset method for simpler equations. We believe with further work and some refinement, that math boxes will be the preferred option for expressions of every complexity.

ACKNOWLEDGMENTS

This work is supported in part by NSF CAREER award IIS-0845921 and NSF award CCF-1012056. We also would like to thank the members of ISUE lab for their support and the anonymous reviewers for their useful comments and feedback.

REFERENCES

1. Anthony, L., Yang, J., and Koedinger, K. R. A paradigm for handwriting-based intelligent tutors. *Int. J. Hum.-Comput. Stud. 70*, 11 (Nov. 2012), 866–887.
2. Blackwell, F. W., and Anderson, R. H. An on-line symbolic mathematics system using hand-printed two-dimensional notation. In *Proceedings of the 1969 24th National Conference*, ACM '69, ACM (New York, NY, USA, 1969), 551–557.
3. Bott, J. N., Gabriele, D., and LaViola, Jr., J. J. Now or later: An initial exploration into user perception of mathematical expression recognition feedback. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, ACM (New York, NY, USA, 2011), 125–132.
4. Jiang, Y., Tian, F., Wang, H., Zhang, X., Wang, X., and Dai, G. Intelligent understanding of handwritten geometry theorem proving. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, IUI '10, ACM (New York, NY, USA, 2010), 119–128.

5. Kang, B., and LaViola, J. Logicpad: A pen-based application for visualization and verification of boolean algebra. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI '12, ACM (New York, NY, USA, 2012), 265–268.
6. Labahn, G., Lank, E., Marzouk, M., Bunt, A., MacLean, S., and Tausky, D. Mathbrush: A case study for pen-based interactive mathematics. In *Proceedings of the Fifth Eurographics conference on Sketch-Based Interfaces and Modeling*, Eurographics Association (2008), 143–150.
7. LaViola, Jr., J. J., and Zeleznik, R. C. Mathpad2: A system for the creation and exploration of mathematical sketches. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 432–440.
8. LaViola Jr, J. J., Leal, A., Miller, T. S., and Zeleznik, R. C. Evaluation of techniques for visualizing mathematical expression recognition results. In *Proceedings of graphics interface 2008*, Canadian Information Processing Society (2008), 131–138.
9. Lee, W., de Silva, R., Peterson, E. J., Calfee, R. C., and Stahovich, T. F. Newton's pen: A pen-based tutoring system for statics. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, SBIM '07, ACM (New York, NY, USA, 2007), 59–66.
10. Li, C., Zeleznik, R., Miller, T., and LaViola, J. J. Online recognition of handwritten mathematical expressions with support for matrices. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, IEEE (2008), 1–4.
11. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U., et al. Icfhr 2014 competition on recognition of on-line handwritten mathematical expressions (crohme 2014). In *Proceedings of International Conference on Frontiers in Handwriting Recognition* (2014).
12. Nakano, Y., and Murao, H. Bredima: Yet another web-browser tool for editing mathematical expressions. In *Proceedings of MathUI* (2006).
13. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.
14. Smithies, S., Novins, K., and Arvo, J. A handwriting-based equation editor. In *Graphics Interface*, vol. 99 (1999), 84–91.
15. Smithies, S., Novins, K., and Arvo, J. Equation entry and editing via handwriting and gesture recognition. *Behaviour & information technology 20*, 1 (2001), 53–67.
16. Tapia, E., and Rojas, R. A survey on recognition of on-line handwritten mathematical notation. *Freie Universitat Berlin, Institut fur Informatik, Germany* (2007).
17. Vintsyuk, T. Speech discrimination by dynamic programming. *Cybernetics and Systems Analysis 4*, 1 (1968), 52–57.
18. Wais, P., Wolin, A., and Alvarado, C. Designing a sketch recognition front-end: User perception of interface elements. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, SBIM '07, ACM (New York, NY, USA, 2007), 99–106.
19. Zanibbi, R., and Blostein, D. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR) 15*, 4 (2012), 331–357.
20. Zanibbi, R., Novins, K., Arvo, J., and Zanibbi, K. Aiding manipulation of handwritten mathematical expressions through style-preserving morphs. In *Proceedings of Graphics Interface 2001*, GI '01, Canadian Information Processing Society (Toronto, Ont., Canada, Canada, 2001), 127–134.
21. Zeleznik, R., Bragdon, A., Adeputra, F., and Ko, H.-S. Hands-on math: A page-based multi-touch and pen desktop for technical work and problem solving. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, ACM (New York, NY, USA, 2010), 17–26.
22. Zeleznik, R., Miller, T., and Li, C. Designing ui techniques for handwritten mathematics. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, ACM (2007), 91–98.
23. Zeleznik, R., Miller, T., Li, C., and Laviola Jr, J. J. Mathpaper: Mathematical sketching with fluid support for interactive computation. In *Smart Graphics*, Springer (2008), 20–32.