MathPaper: Mathematical Sketching with Fluid Support for Interactive Computation

Robert Zeleznik[1] and Timothy Miller[1] and
Chuanjun Li[1] and Joseph J. LaViola Jr.[2]

[1] Brown University
[2] University of Central Florida

Abstract. We present MathPaper, a system for fluid pen-based entry and editing of mathematics with support for interactive computation. MathPaper provides a paper-like environment in which multiple mathematical expressions and even algorithms can be entered anywhere on the page. Mathematical expressions can also be modified using simple deletion and dragging gestures with real-time recognition and computation feedback. In addition, we support extended notations and gestures for controlling computational assistance, simplifying input, and entering algorithms, making MathPaper a user-friendly system for mathematical sketching and computation.

1 Introduction

Despite the tremendous computational power that is available on a typical desktop, it is interesting that students and professionals frequently resort to pencil and paper to do their homework or solve problems. Several factors conspire toward this phenomenon, such as the time it takes to start up an application and the limited display surface of a computer screen. However, we believe that the dominant reason for this reluctance to utilize desktop mathematics applications is the nature of pencil and paper itself, which affords fluid, direct specification of 2D mathematical notations and supporting diagrammatic structures in arbitrary juxtapositions.

From this motivation, we have developed MathPaper, a virtual equivalent to pencil and paper, that is enhanced to provide integrated computational support for mathematics. Ideally this should be at least as convenient as writing on paper but enhanced to recognize mathematical structures including multiple expressions, matrices, and even algorithms. Additionally, given the complexity of general mathematics recognition, including inherent notational ambiguities, we believe that interactive recognition feedback must be provided to ensure a fluid workflow. Previous systems [1, 2] that require explicit interaction to display recognition feedback after an expression has been entered necessarily introduce a heavyweight cognitive step — users must shift from thinking about their problem at hand to examining the subtle symbolic and spatial aspects of an input expression to see if it matches their intentions. Instead, when provided with interactive recognition feedback, we believe users can amortize the inspection of

recognition feedback over the course of their input. Changing to this lighter-weight process may make the essential nature of pencil-and-paper as a fluid, non-disruptive medium attainable.

Recognizing mathematical expressions non-disruptively, however, is only the first step in creating MathPaper. Additionally, we need extensions to standard mathematical notation to support computational assistance. Believing that mathematical notations are rich, familiar, and expressive, we attempted to introduce mathematical assistance in the form of enhanced mathematical notations. Thus, instead of treating computation as an external method to apply to a handwritten expression, we treat computation as an integral part of math notation. For example, drawing a $\Rightarrow$ to the right of any expression computes a numerical approximation of that expression. Although eventually some kind of computationally extended notation might be accepted as a standard that everyone would learn in school, we, in the meantime, need to provide additional support both for learning notational extensions and for accessing functionality that is more conveniently accessible through other user interface mechanisms.

In the next section of this paper, we discuss work related to the MathPaper prototype followed by a discussion of the fundamental interactions it supports. We then discuss how mathematics is specified and recognized in the MathPaper system and present some details on its support for computation. Finally, we present a brief discussion on an informal user evaluation along with future work and conclusions.

2 Related work

Sketch-based interfaces have been explored for facilitating conceptual designs in various fields, including 2D diagrams [3] by Gross et al., mechanical design [4, 5] by Alvarado and Kara, musical score creation [6] by Forsberg et al., 3D scenes [7] by Zeleznik et al., and freeform models [8] by Igarashi et al. In contrast, MathPaper provides a sketch-based interface for mathematical expression input, editing and computation.

For mathematical computation, sketch-based interfaces have also been developed by quite a few people and companies. Chan and Yeung developed a simple pen-based calculator PenCalc [9], and Thimbleby and Thimbleby also developed a calculator with a gesture-based interface supporting animation for simple math calculations [10]. MathBrush [11] recognizes mathematics and drives a symbolic algebra system with it. MathJournal^{TM}by xThink, Inc. can solve equations, perform symbolic computation, make graphs, and automatically segment multiple expressions, while Microsoft Math^{TM} [1] supports both numeric and symbolic computation of a single expression. LaViola's MathPad^2 system [2] used a pen-based interface to let users create dynamic illustrations for exploring mathematical and physical concepts by combining handwritten mathematics and free-form drawings. Comparing with those sketch-based interfaces for mathematics, MathPaper provides a real-time, write-anywhere, fluid interface for mathematical computations. Expression recognition is automatic, with au-

tomatic segmentation for multiple expressions. Ink stroke deletion and dragging are supported with gestures, and matrices, including non-well-formed matrices, are supported together with algorithm sketches.

3 Fundamental interactions

The fundamental interactions provided by MathPaper are for entering and editing mathematical expressions, issuing commands, and demonstrating the gestural user interface. With these interactions, our design philosophy was to ensure a high degree of real-time interactivity and feedback both at the recognition level and at the computation level (as in spreadsheets where changing different values automatically affect the results of various formulae).

3.1 Ink input and editing by handwriting

Mathematical expressions are entered by writing virtual "ink", just as with paper. As Fig. 1 shows, multiple expressions are supported, and except for the cases where orders of expressions are important for computation, dependent expressions can be entered anywhere on the page, just like on paper. The right arrow is an extended notation for expression computation, detailed in Section 4.1.

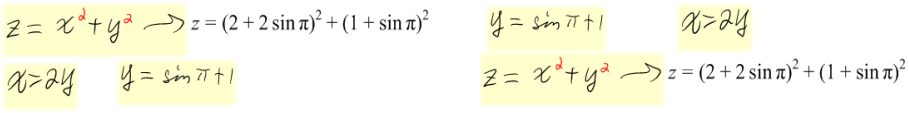

Fig. 1. Arrangements of dependent expressions. The expressions on the left are the same as on the right, yet arranged differently.

Editing of entered ink strokes can be done by circling and then dragging them or by scribbling over them to delete them; recognition of both gestures is discussed in Section 5.3. In addition, the horizontal bar of a square root or fraction can be extended by just drawing another horizontal line in the same direction with one end near the end of the line to be extended.

3.2 Commands

Since we provide more commands in our system than we can provide easy gestural or notational shortcuts for, we make use of widgets attached to each written expression to provide additional functionality. However, adding a normal widget would take up screen space, preventing the user from entering ink in that location. To conform to our write anywhere aesthetic, we avoid this problem by using a technique inspired by hover widgets [12]. We draw a green square under the ink on the display; hovering the pen tip over the square without touching the screen causes a cluster of buttons to appear to the upper-right (see Fig. 2). If the pen moves directly towards the buttons, they can be pressed and interacted with as usual to display menus, perform functions, etc. If the pen moves in any

other direction first, the buttons are removed. Thus, the user may draw strokes at any location on the screen and still have access to local widgets. If the user happens to move accidentally over the buttons, it is a simple matter to move back off again, although, in practice, this rarely occurs.

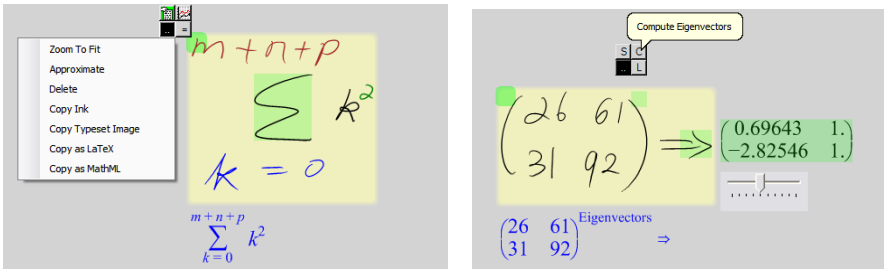

Fig. 2. Embedded widgets associated with specific expression locations.

A small green square is provided at the top left of each expression to support various expression level commands such as graphing and simplification. For matrices, a green square is also placed at the top right with matrix-specific operations such as computation of eigenvectors, singular value decompositions, and norms. (The green boxes in figures other than Fig. 2 have been deliberately set to have the same color as the background to reduce distraction in this paper).

Demonstrations of extended notations and gestures We attempt to make some of the extended notations and gestures we provide (see Section 4.1) discoverable by demonstrating to the user what the gesture would have been to perform a command the user has selected. For instance, if the user selects Delete from the menu, the system draws a scribble as an animation over the ink before deletion. Currently this happens for the Delete, Simplify, Numerically Approximate, and Graph commands. In the case of the notations for Simplify and Numerically Approximate, the ink for the notation is left on the screen as if the user had written it, since these are intended to be persistent marks.

3.3 Interaction for switching between drawing and math modes

In addition to mathematical expressions, MathPaper also supports drawing of freeform shapes and/or annotations without recognition or interpretation. Mode switching between math mode and drawing mode can be done either by choosing a command or by flicking a diagonal line followed by drawing a circle on the line. Ink in the freeform drawing mode is colored in blue. Annotations can also be later recognized as mathematical expressions; this is done by lassoing the ink strokes in the drawing mode followed by drawing a line entering the circle as shown in Fig. 3.

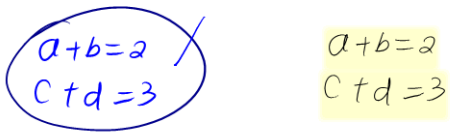

Fig. 3. Recognition of an annotation as math by lassoing the ink followed by an intersecting or crossing line stroke as shown on the left. The recognized strokes are shown on the right.

4 Specifying Mathematics

Our system currently supports entry of a solid range of basic math: a number of basic math symbols (including Greek letters and others such as ∞, $\mathbb{R}$, etc.), basic math relations ($<$, $\leq$, $\neq$, $\approx$, $\supset$, $\perp$, etc.), basic math operators ($+$, $-$, $\cdot$, $/$, $\wedge$, etc.), fractions written vertically ($\frac{a}{b}$), integrals, summations, roots, trigonometric and similar (log, ln, etc.) functions, and matrices. The recognized expressions can be exported as LaTeX code, MathML, and vector images of the typeset display.

Mathematics already uses duplicate notation for multiple purposes. For example, a textbook one author has uses u_x and u_y for the partial derivatives of a function u with respect to x and y, while other authors use that notation to represent the x and y coordinates of u. Perhaps the most common example is the use of each of i and j to represent either $\sqrt{-1}$ or an ordinary variable. Since, as a result, a mathematics recognition system already has to make choices when interpreting notation, it may as well consider recognizing other additions to standard notation to control further processing of the expressions, to take advantage of the pen's strengths to simplify entry, and for various other purposes.

4.1 Extended mathematical notations

The notational extensions discussed here extend users' control of their input, summarized in Table 1.

MathPaper supports interactive computation and graphing, both with arrow notations. A double right arrow at the end of an expression tells the system to compute a numerical approximation, while a single right arrow results in an exact symbolic evaluation, as shown on the left of Fig. 4. Since the only use we make of double arrows is for this numerical approximation function, an "arrow" referred to later in this paper is a single arrow unless otherwise specified.

Arrows are not only used for computation and graphing, but also for input brevity of matrices with patterns as shown on the right of Fig. 4. If a matrix has a certain pattern, input of the matrix can be simplified by extending a diagonal arrow for specifying the pattern. The arrow starts from the element on the first column and the first row, and points to the element on the last column and the last row. As shown on the right of Fig. 4, the dimensions of the matrix can be specified in the bounding box of the arrow, and the row index and column index variables can also be specified along with the dimensions. The number of rows

Table 1. Extended Notations

Functions	Notations	Descriptions
Computation	$\rightarrow, \Rightarrow$	Evaluates expressions
Graphing	$\leftarrow\rightarrow\uparrow\downarrow\nearrow\searrow\nwarrow\swarrow$	Graphs an expression
Brevity	$\searrow$ in matrices	Facilitates input of matrices with patterns
Space Management	$\mid - \neg$	Adds more horizontal or vertical spaces in matrices
Algorithmic Notations	$\leftarrow$	for function definition and return
	$\forall$	a shortcut for *for*
	$\in$	for argument type specification and loops
	$\sim$	for omitted data. Order is enforced.
	$=$	for assignment and equality
	//	for comments; can cover multiple lines
	$//T()$	produces a table of trace results
	$\nearrow\searrow\longrightarrow$	sets a trace point if in function definition

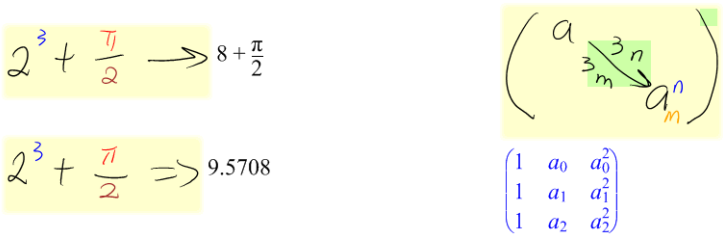

Fig. 4. Examples of extended notations.

and the row index variable are below the arrow, and the number of columns and the column variable are above the arrow. If elements above the main diagonal are a constant value, that value can be entered above and to the right of the arrow bounding box. Similarly, if elements below the main diagonal are constant, that value can be entered below and to the left of the arrow bounding box. This arrow extension allows for quick input of large matrices with certain patterns.

MathPaper also supports gestures for managing spaces between matrix elements. A vertical line that is at least twice the average height of the element bounding boxes moves the strokes on the right of the vertical line to the right a certain distance to allow insertions in the matrix. Similarly, a horizontal line that covers at least two matrix elements moves the strokes below by a certain distance. Since the long horizontal line can only add vertical space to more than one matrix element, we also use a $\neg$ gesture for adding more vertical space to any stroke below the horizontal part of the $\neg$. If any stroke is moved to the right, and the matrix has a closing parenthesis, the closing parenthesis is moved to the right accordingly. If a stroke intersects the space adding gesture, it is moved only if its center is to the right or below the gesture. For a multiple stroke symbol, movement of any stroke also moves all the other strokes of the symbol. These space adding gestures provide a faster way to move strokes than lassoing and dragging them if only horizontal or vertical space is needed.

Besides the above, MathPaper also supports extended notations for algorithm sketching [13] as listed in Table 1. Algorithm Sketching in MathPaper supports sketched algorithmic pseudocode with online computation capabilities. It supports flow of control constructs such as *for* loop, *if*, and *else*, and also supports a trace table for showing values of variables at any running point. A left arrow following function name with parameters specifies a function definition, while a heading left arrow returns function call results. For input simplicity, notations are extended for pen-based input. For example, $\forall$ is used for *for*, $\sim$ is used for ellipsis $(\cdots)$. Mathematical notation $\in$ is also supported for specifying argument types and ranges of iteration variables in *for* loops. The trace table is specified by the table head $//T()$, with trace variables listed in the parentheses. Right and diagonal arrows are extended for specifying the trace point in the algorithm, with the arrow starting from the trace point, and pointing to the trace table head. Comments can be added following //, and the extended notation // can cover multiple lines, rather than only one line as used for keyboard and mouse input.

By extending notations, pen-based input can be simplified, and interactions can be more fluidly supported. All extended notations have specific meanings depending on context.

4.2 Allograph mapping

MathPaper explicitly represents allographs, or different ways of writing the same symbol, with different user-specifiable interpretations for different allographs even when they are of the same symbol. As in our previous work [14], this means that, for instance, a script i could be mapped to the variable i while a straight-line i could be simultaneously mapped to $\sqrt{-1}$.

5 Recognizing mathematics

Fig. 5 shows the major steps involved in the math recognition used in MathPaper. After recognition of an input symbol, affected ranges (each range represents one grouping of symbols that the system believes is a single complete expression) are re-parsed automatically. Symbols in one range are sorted and a baseline tree is constructed for the range. After a semantics parse step, an expression is generated for one affected range. More details about the recognition process are given in [13].

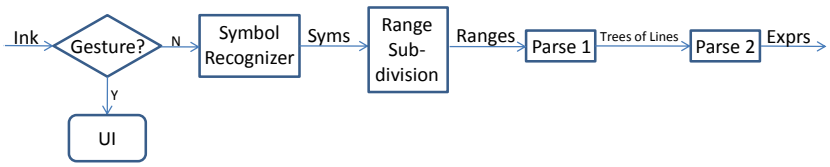

Fig. 5. Dataflow of mathematical expression recognition from ink strokes(from [13]).

5.1 Interactive feedback

MathPaper re-recognizes and re-parses after each update, including stroke entry or deletion, as well as dynamically in real-time during interactive movement. If a new stroke is added, it is recognized, and all the symbols in the affected range are sorted and parsed again with the immediate feedback as the typeset output. If more than one range is affected, for example by deleting a stroke which splits one range into two new ranges, or by adding a stroke which merges multiple ranges, all affected ranges are re-sorted and re-parsed.

Extended typeset conventions The recognition results are generally displayed as 2D typeset expressions under the input strokes for each range, although there are alternate styles available [14]. Our typeset display algorithm has evolved to be very close to implementing the basic TEX display algorithm [15] with some additional knowledge about conventions that TEX relies on the user to follow. This feedback is only displayed under the expression the cursor was last over, to avoid confusion and clutter. One change we have made is to use a different typeface to distinguish the variable representing $\sqrt{-1}$ and the base of the natural logarithm each from the corresponding ordinary variables with the same symbols (i.e., i vs i and e vs e, respectively).

Fig. 6. Typeset convention extensions.

In addition, we have explored extensions to indicate syntax errors by highlighting the relevant part of the output typeset with a red squiggle underneath it, as in Microsoft Word's indication of spelling mistakes. In the case of roots and vertical fractions we have found it more visually comprehensible to indicate a missing value by two bold question marks as shown in Fig. 6. In order to provide this feedback, our parser parses even mathematically invalid expressions, adding special nodes to the internal parse tree to indicate values that were assumed to be missing.

5.2 Automatic range segmentation

MathPaper supports multiple expressions, each represented by a range, and ranges are automatically segmented according to spaces between strokes, sizes and identities of the symbols at the boundaries of neighboring ranges.

After a stroke is recognized as a symbol, an inflated bounding box is computed for use in range identification. The amount to be inflated in each direction is based on the size, shape, and identity of the symbol. Special cases such as comma (,) and +/ =>< etc. are prevented from becoming isolated ranges even if they have small bounding boxes and relatively large distances to ranges on both sides.

After computing the inflated rectangle r for range identification, all existing ranges are examined against r using intersection tests. If an existing range's bounding box intersects r, the update belongs to the existing range. Otherwise, the existing range's bounding box is inflated. Let the inflated bounding box of the existing range be R. If R does not intersect r, the update does not belong to the existing range. If R intersects r, the bounding box of the nearest symbol in the range is inflated and if the inflated bounding box intersects r, the update hits the range. Otherwise the range is not hit by the update.

Hence, update is automatic not only when there is a single expression, but also when there are multiple expressions. Range segmentation is automatic after each update. After identifying all the hit ranges, symbols in them are re-sorted and parsed for the updated expressions.

5.3 Gesture disambiguation

Gestures disambiguation is done in MathPaper by examining the contexts of the gestures. A stroke is recognized as a deletion gesture only if it intersects other strokes. As discussed above, lasso gestures are used to drag ink strokes around, and a stroke is recognized as a lasso only if it contains at least one stroke. Also for tapping as a gesture for the integral sign and summation sign upper limit and lower limit widget [14], it can be recognized as a tapping gesture only if it is in the bounding box of an integral or summation symbol. If a tapping is followed by another stroke which will make a symbol i or j, or a tapping can make a symbol i or j with a neighboring stroke, it is recognized as a symbol rather than a tapping gesture even if the tapping is inside the bounding box of an integral or summation symbol. Similarly, a single arrow can be used for symbolic computation, graphing, algorithm definition, etc, and its semantic meaning depends on its contexts. More detail on graphing gestures is found in Section 6.3.

6 Supporting computation

Symbolic computation in our system is supported through mathematics engine plug-ins. Currently, only two are fully functional: one that uses Mathematica [16] as a back-end to do the computations, and the other, which supports a limited but still useful set of calculations, was developed in house.

6.1 Automatic update

One of MathPaper's key features is its support of automatic updates for computations. If there is a change to an assignment expression, all computations are updated automatically as shown in Fig. 7 for the two computations of $a + b$ and $a^2 + b^2$ after the update of $c = T$ into $c = \pi$. Although the updated assignment might not be in a computation as shown for the update of $c = \pi$ in Fig. 7, this assignment update might affect other assignments, hence all computations need to be updated with any assignment update in order to have automatic computation updates.

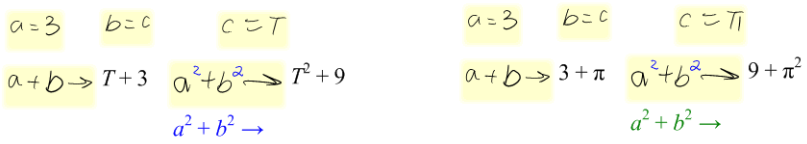

Fig. 7. Automatic update of computations after ink $c = T$ shown on the left is updated to $c = \pi$ shown on the right.

6.2 Display feedback and interaction with computation results

If a computation result is asked for by the user, MathPaper attempts to determine if the result is a number. If the system is sure that the result is a number (whether exact or approximated), it displays the typeset feedback for the input expression in green rather than blue (see Fig. 7).

MathPaper also supports interaction with computation results if the results are numerical approximations. Numerical results are highlighted in green, and hovering over the results displays a sliding bar below the highlighted area as shown on the right of Fig. 2. The sliding bar is used to interactively change the desired number of decimal places that numerical results should be rounded to. An alternate option to the sliding bar is to use a horizontal stroke directly, or just hover over the results to specify the decimal places to round off.

6.3 Graphing

In addition to computation, MathPaper also supports graphing of expressions with an arrow gesture. If an arrow starts from within the bounding box of an expression, and points to outside of the bounding box of any expression, it is a graphing gesture. The graph will be plotted at the arrow head with the arrow ink being deleted. If a graphing arrow goes through multiple expression bounding boxes, all touched expressions will be plotted in one graph as shown in Fig. 8. MathPaper uses arrows rather than curved lines as used in MathPad^2 [2] for graphing to avoid a parenthesis being mis-recognized as a graphing gesture. To disambiguate a right arrow for symbolic evaluation from a graphing right arrow, a graphing right arrow must intersect at least one stroke in the expression, otherwise it will be interpreted as a computation gesture.

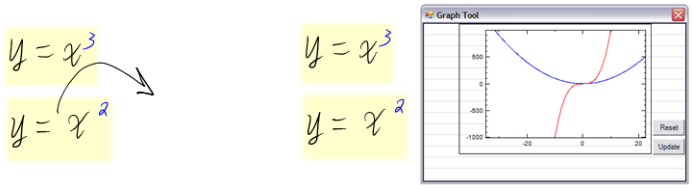

Fig. 8. Graphing of multiple expressions by one arrow.

7 Discussion

We conducted an informal usability evaluation of MathPaper by providing the prototype to seven teachers of mathematics and science at both the high school and college levels. Along with the software, the teachers were also given an instruction manual and they were asked to experiment with MathPaper, focusing on entry, simplification, and graphing of simple mathematical expressions commonly found at the Algebra I level.

The majority of the teachers found MathPaper's interface fairly straightforward and easy to use and found the real-time recognition feedback useful in assisting them with correcting recognition mistakes. In addition, they found the real-time computation feedback, when modifying existing variables and expressions, to be an important part of MathPaper's functionality, since it let them quickly examine the results of any changes they made. They also reported that this feature is especially useful when teaching students how changes in different parts of a mathematical expression affect a computational result. The majority of the teachers found the graphing functionality to be somewhat lacking in terms of the types of graphs it supports but found its gestural interface straightforward. In total, the teachers enjoyed using MathPaper but also felt that it needed more mathematics computation functionality.

8 Future Work

MathPaper provides a fluid user interface for automatic recognition of multiple mathematical expressions and for mathematical computations. Currently, the user must explicitly switch modes between interpreted math and un-interpreted drawing. It would be more fluid to not have this drawing mode by relying on automatic detection of annotations or freeform drawing.

We have supported editing of ink strokes only, with no editing allowed for the typeset displayed. However, some MathPaper users have tended to try to edit the typeset directly. In the future, we would like to explore schemes for allowing editing of the typeset without taking available screen space away from starting new expressions.

Displaying of computation results has fixed locations and computation results cannot be reassigned to new variables. Being able to drag the results and to assign the results to new variables would be obviously quite helpful. Expression evaluation is supported, yet solving an equation for a specific variable is also important and should be supported in the future. Finally, a formal usability study is being developed for MathPaper that will explore how MathPaper can be accepted by users and how its UI can provide them with a more productive and enjoyable experience.

9 Conclusion

We have presented MathPaper, a mathematical sketching system for interactive computation. MathPaper supports automatic recognition of both mathematical

expressions, including well-formed and non-well-formed matrices, and sketched algorithmic pseudocode, and automatic segmentation of multiple expressions. Expressions can be entered anywhere on the page, and ink strokes can be deleted, dragged and modified for error correction and expression re-arrangement. Mathematical notations have been extended and unambiguous gestures have been designed in MathPaper to support fluid interactions with mathematics.

Acknowledgements

This work is supported, in part, with grants from Microsoft Research and IARPA.

References

1. Microsoft: Math. Computer program. `http://www.microsoft.com/math`.
2. LaViola, J., Zeleznik, R.: Mathpad^2: A system for the creation and exploration of mathematical sketches. ACM Transactions on Graphics **23**(3) (August 2004) 432–440 (Proceedings of SIGGRAPH 2004).
3. Gross, M.D., Do, E.Y.L.: Ambiguous intentions: a paper-like interface for creative design. In: UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology, New York, NY, USA, ACM (1996) 183–192
4. Alvarado, C.: A natural sketching environment: Bringing the computer into early stages of mechanical design. Technical report, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (May 2000)
5. Kara, L.B., Gennari, L., Stahovich, T.F.: A sketch-based interface for the design and analysis of simple vibratory mechanical systems. In: Proceedings of ASME International Design Engineering Technical Conferences. (2004)
6. Forsberg, A., Dieterich, M., Zeleznik, R.: The music notepad. In: UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology, New York, NY, USA, ACM (1998) 203–210
7. Zeleznik, R.C., Herndon, K.P., Hughes, J.F.: Sketch: an interface for sketching 3d scenes. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM (1996) 163–170
8. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: a sketching interface for 3d freeform design. In: SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press/Addison-Wesley Publishing Co. (1999) 409–416
9. Chan, K.F., Yeung, D.Y.: Pencalc: A novel application of on-line mathematical expression recognition technology. In: Proceedings of the Sixth International Conference on Document Analysis and Recognition. (September 2001) 774–778
10. Thimbleby, W., Thimbleby, H.: A novel gesture-based calculator and its design principles. In MacKinnon, L., Bertelsen, O., Bryan-Kinns, N., eds.: Proceedings 19th BCS HCI Conference. Volume 2., British Computer Society (2005) 27–32
11. Labahn, G., MacLean, S., Mirette, M., Rutherford, I., Tausky, D.: Mathbrush: An experimental pen-based math system. In Decker, W., Dewar, M., Kaltofen, E., Watt, S., eds.: Challenges in Symbolic Computation Software. Number 06271 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
12. Grossman, T., Hinckley, K., Baudisch, P., Agrawala, M., Balakrishnan, R.: Hover widgets: Using the tracking state to extend the capabilities of pen-operated devices. In: Proceedings of CHI 2006. (April 2006) 861–870
13. Li, C., Miller, T., Zeleznik, R., LaViola, J.: Algosketch: Algorithm sketching and interactive computation. In: Proceedings of the 5th EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (SBIM 2008). (June 2008) 175–182
14. Zeleznik, R., Miller, T., Li, C.: Designing UI techniques for handwritten mathematics. In: Proceedings of the 4th EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (SBIM 2007). (August 2007)
15. Knuth, D.E.: The TEXbook. Addison Wesley (1986)
16. Wolfram, S.: Mathematica: A System for Doing Mathematics by Computer. Addison-Wesley (1988)