Aiding Manipulation of Handwritten Mathematical Expressions through Style-Preserving Morphs

Richard Zanibbi
Department of Computer Science
Queen's University
Kingston, Ontario, Canada
zanibbi@cs.queensu.ca

Kevin Novins
Department of Computer Science
University of Otago
Dunedin, New Zealand
novins@cs.otago.ac.nz

James Arvo
Department of Computer Science
California Institute of Technology
Pasadena, California
arvo@cs.caltech.edu

Katherine Zanibbi
Department of Psychology
Queen's University
Kingston, Ontario, Canada
zanibbi@psyc.queensu.ca

Abstract

We describe a technique for enhancing a user's ability to manipulate hand-printed symbolic information by automatically improving legibility and simultaneously providing immediate feedback on the system's current structural interpretation of the information. Our initial application is a handwriting-based equation editor. Once the user has written a formula, the individual hand-drawn symbols can be gradually translated and scaled to closely approximate their relative positions and sizes in a corresponding typeset version. These transformations preserve the characteristics, or *style*, of the original user-drawn symbols. In applying this *style-preserving morph,* the system improves the legibility of the user-drawn symbols by correcting alignment and scaling, and also reveals the baseline structure of the symbols that has been inferred by system. We performed a preliminary user study that indicates that this new method of feedback is a useful addition to a conventional interpretive interface. We believe this is because the style preserving morph makes it easier to understand the correspondence between the original input and interpreted output than methods that radically change the appearance of the original input.

Key words: formula entry, math recognition, mental map, morphing, pen-based computing, user feedback.

1 Introduction

Traditional interfaces require the user to enter input in an unambiguous machine-readable form. While this approach is highly effective for text and numeric entry, it can become a substantial burden in the the case of diagrammatic input. For rough sketches, users often prefer to explore their ideas using pieces of scrap paper rather than powerful software packages [1]. Recently, researchers have begun working to retain the appeal of pencil and paper by adding interpretive layers to their interfaces [2, 3, 6, 7, 11, 14, 15]

Automatic interpretation will always be subject to mistakes. An essential feature of an interpretive interface is therefore to provide feedback to the user and to allow for correction of the inevitable recognition errors. Success of the interface depends on three factors: (1) the accuracy of interpretation, (2) the quality of the feedback, and (3) the ease of error correction.

High quality feedback conveys information clearly without undue disruption. In this paper, we focus on the problem of providing unobtrusive feedback from a recognition engine. We chose the entry and editing of mathematical expressions using handwritten input as an application area. We use this context to introduce the notion of style-preserving morphs, which are gradual transformations of the user's input that communicate information about the recognition process without obliterating the user's writing style. We hypothesize that stylistic cues can be helpful in maintaining the user's mental map of the input.

We performed a preliminary user study that compares three versions of a handwriting based equation editor – one with a conventional typeset feedback mechanism, one with a style-preserving morph, and one with both. We found that equation entry times were statistically indistinguishable with the first two interfaces. Most users felt that the style-preserving morph was a useful addition to a conventional interface.

2 Style-Preserving Morphs

Pen-based computer systems such as the Apple Newton and the Palm Pilot provide feedback by instantaneously replacing handwritten input with a typeset interpretation [10]. Often, the typeset version is displaced from the original input. Such changes in layout can seriously disrupt a user's *mental map*, which is an essential ingredient of inferred semantics. Within the graph drawing community, for example, it has been observed that when a user's mental map of the information is degraded, as when the drawing is changed, subsequent understanding of the drawing is hindered [9, 12].

In order to minimize disruption of the user's mental map, researchers have recently explored the use of morphing to provide smooth transitions of raw input to clean typeset representations [2, 3]. By forcing all changes to be gradual, the user can easily keep track of what is happening. Surprisingly, introducing inertia into the system makes the interface feel more comfortable despite the reduced response time.

Transformation to a clean typeset representation provides useful feedback from an interpretation process. Unfortunately, it also has the effect of destroying the character of the original input. This is unfortunate since users have been shown to prefer rough-looking sketches during the design phase of a project [5, 7]. They state that clean typeset output connotes authority and immutability. Systems that output CAD models using a hand-drawn style [7, 11] and systems that preserve hand-drawn strokes via reprojection [15] are emerging.

In a style-preserving morph, we restrict feedback to the gradual repositioning and scaling of individual input strokes. Handwritten input is still easily recognizable after such transformations (see Figure 2). Yet they are powerful enough to provide feedback from the recognizer to the user on its sense of which symbols should be aligned, and which symbols should be the same size.

3 Interpretation of Handwritten Equations

We chose online handwritten equation entry and editing as an application domain in which to examine the feasibility of the style-preserving morph. This domain is attractive in that the 2D nature of mathematical notation makes it a prime candidate for freehand input [14].

Our experimental prototype for online equation entry is an extension of Smithies et al.'s Freehand Formula Entry System (FFES) [13, 14]. This system combines online character recognition and parsing with a graphical user interface that contains modules for feedback and correction of interpretation errors at the level of individual symbols. FFES converts handwritten input into LaTeX notation and can display a typeset result.

$$X^{a+b_2} + \frac{2}{3}$$

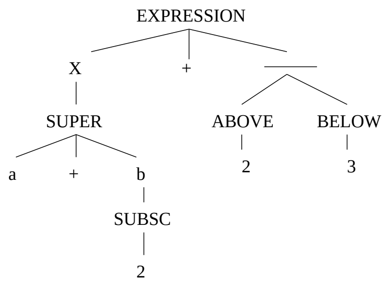


Figure 1: An expression and its baseline structure tree.

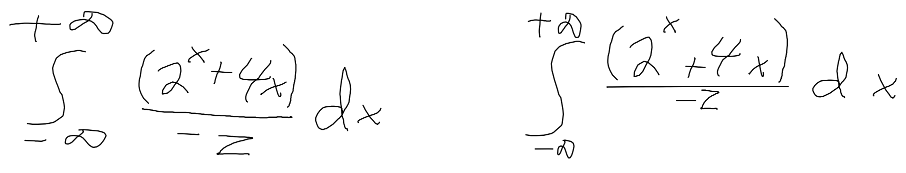
FFES' major weakness is that its graph rewriting parser is slow and unreliable [14]. Parsing can take tens of seconds, and in the case of input that is outside the range of FFES' grammar, the only feedback is the largest grammatically correct unit that the parser was able to locate.

We have replaced the original parser with the Diagram Recognition Application for Computer Understanding of Large Algebraic Expressions (DRACULAE) [16, 17, 18] implemented in the tree-rewriting language TXL [4]. DRACULAE takes as input a set of symbols with bounding boxes and produces a *baseline structure tree* which describes the hierarchical decomposition of baselines in an expression (see Figure 1). Symbols sharing a baseline are represented as left-to-right ordered siblings below a node labeled with the baselines' associated region. EXPRESSION represents the region containing an entire expression, while the remaining region labels represent a region relative to their parent symbol node. Complex formulas can be processed by DRACULAE in under a second on a 200MHz machine. Also, DRACULAE always produces a baseline structure tree containing all input symbols, making it possible to provide feedback to the user even if an expression has syntax errors. DRACULAE can translate a baseline structure tree to TEX , or to an operator tree [16, 17].

4 Morphing Algorithm

We have introduced a style-preserving morph into FFES as part of a new operation, which we call *Align*. An example of the result of the Align operation is shown in Figure 2. The symbols of the expression are aligned on detected baselines and resized for consistency. Note that the input writing style has been preserved.

The style-preserving morph result provides useful feedback on symbol layout in most cases. Errors are of-



a. Original input b. After style-preserving morph

Figure 2: The effect of a Style-Preserving Morph.

ten obvious, for example in Figure 3, where the superscripted "2" has been interpreted as being adjacent to the "x". Once these types of errors are identified, they are easy to fix using operations built into FFES.

In addition to increasing consistency, the resizing of symbols also provides a small amount of feedback on character recognition: if a handwritten symbol is squashed or stretched into an unexpected shape, it is a sure sign of a recognition error. For example, in Figure 3, the lower-case "x" has clearly been mis-recognized.

The new bounding boxes for symbols after the morph are computed bottom-up from the baseline structure tree returned by DRACULAE, proceeding from leaf baselines to the baseline below the EXPRESSION node at the root of the tree.

The procedure for formatting the bounding boxes of each baseline is as follows:

1. An average of the height of the baseline symbols is computed, where each symbol height is weighted based on identity. For example, the height of a lower-case letter such as "x" is doubled. Flat symbols such as horizontal lines are not included in the average. Average height and typeset symbol aspect ratios are then used to recompute the bounding box sizes.

2. Baseline symbols are vertically aligned and any subexpressions nested vertically relative to a symbol (e.g. superscripted, below) are placed 15 pixels above/below the parent symbol.

3. Horizontal spacing is cleaned up: first subexpressions which are nested above or below baseline symbols are centered, and super and subscripted subexpressions are spaced at a fixed distance of 15 pixels to the right of their parent symbol. Then an average of the baseline symbol widths is computed (thin symbols such as 'i' are not included in the average), and the larger of 15 pixels and 1/3 the average width is placed between the areas filled by each baseline symbol and its nested subexpressions.

a. Original input b. After style-preserving morph

Figure 3: Interpretation errors become clear.

The Align operation must preserve the semantics, which implies that aligning input should not alter baseline structure. Ensuring that DRACULAE would produce the same baseline structure tree before and after using Align was one of the key design constraints for the formatting algorithm.

The formatting algorithm executes in under a second on a 200MHz machine. Once it is complete we have a target bounding box for each of the user-drawn symbols. The source and target bounding boxes are used to define a geometric transformation comprised of a translation and a scale that will achieve the desired effect. The transformation is divided into equal steps that are applied in sequence to produce a morphing effect. We implemented the morph in Tcl/Tk. It is achieved in fifteen frames, displayed at approximately ten frames a second. In comparison, a typeset representation can be generated and displayed in under a second.

5 Experiment

Our enhanced version of the equation editor could provide feedback in terms of a conventional Render operation (resulting in a typeset formula being displayed as a bitmap in a separate window) or an Align operation (resulting in a style-preserving morph on the user's drawing canvas). In an attempt to isolate the effect of the style-preserving morph, we designed an experiment to compare the performance of FFES under three conditions of available feedback:

1. conventional Render operation but no Align operation,

2. Align operation but no Render operation, and

3. both Render and Align operations.

In each case the user was asked to enter formulae and to apply edit operations until the system correctly interpreted each one. We measured total time taken to enter each formula so that it was interpreted correctly and counted feedback requests.

A Render operation gives more information than Align, since it unambiguously displays symbol interpretations. It also runs considerably faster than Align, because the morph has a built-in lag. For these reasons, we hypothesized that (1) Render would be more effective in communicating the system's interpretation and (2) participants would enter and correct expressions more quickly using Render than with Align. Nonetheless, we expected subjects to feel more comfortable with Align-style feedback.

5.1 Method

Participants

There were 27 participants in the experiment, five female (18.5%) and 22 male (81.5%), with a mean age of 28.92 years (SD=7.54). Nineteen participants were graduate students (14 Computing Science, three Psychology, two Math), four were Computing Science professors, three were Computing Science undergraduates and one was a software developer.

Materials

Materials consisted of a data tablet attached to a 900 MHz Linux machine with 256MB of RAM running FFES, and a questionnaire (see Appendix A). FFES is a pen-based system; as a result no keyboard or mouse was used in the experiment.

FFES had to be modified to collect the data we needed for the experiment. A screen shot of an experiment in progress is shown in Figure 4. The user drawing canvas dominates the screen. The panel on the lower right displays the target formula to be entered. The panel on the lower left is used to display the results of a Render operation, if enabled.

The expressions used in the experiment are shown in Table 1. They are designed to be of increasing complexity, with expressions three and four being of roughly equal complexity. We gauged the complexity of expressions using number of symbols and number of symbols which are subscripted, superscripted, or above or below another symbol.

1.	$n(1-a)$
2.	$y = \frac{3^{x+1}}{5x} + z$
3.	$\frac{1}{N-1}\sum_{k=1}^{N}(t_k - m)^2$
4.	$\int_{-\infty}^{+\infty} \frac{(2^x+4x)}{-z} dx$

Table 1: Expressions used in the experiment. Expression four was entered only in the final condition (FFES with Render and Align).

Procedure

After completing a consent form, all participants were given a scripted walk-through of the operations of FFES, and introduced to the three experimental conditions: (1) FFES with Render only (2) FFES with Align only and (3) FFES with Render and Align. The participants then performed two practice trials, entering $E = mc^2$ in the Render-only and Align-only conditions with the experimenter available for questions.

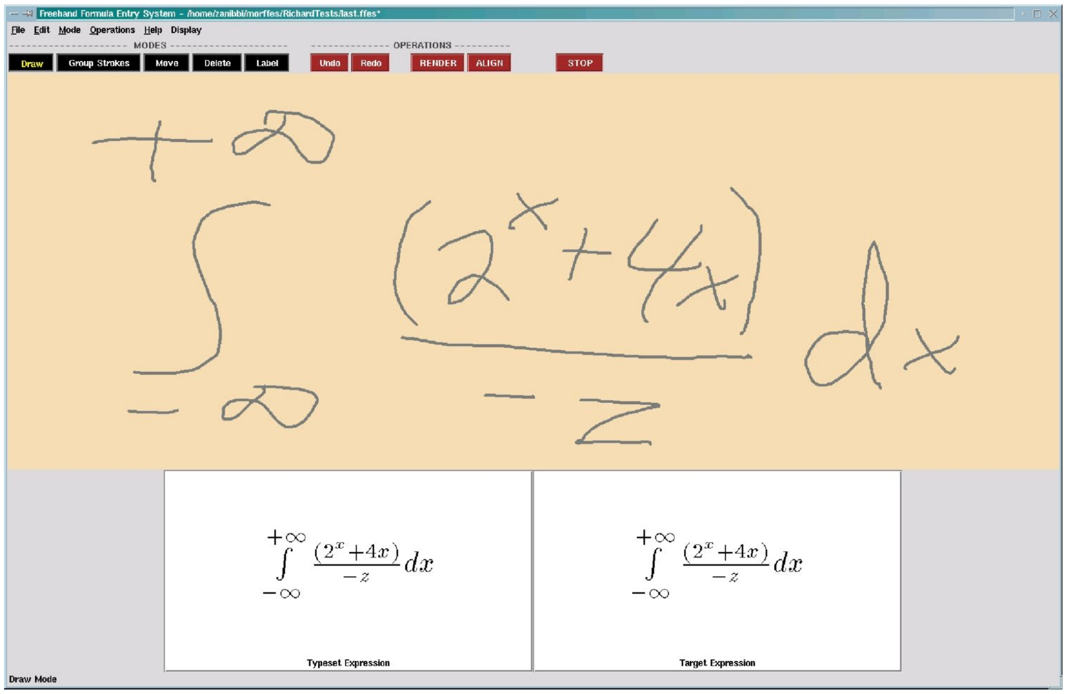
For each trial the participants first hit a start button to view the target expression (displayed in the bottom right panel of FFES, see Figure 4). When the participants felt that the system had correctly interpreted their input, they pressed a button labeled "Stop". If Stop was pressed and the last interpretation did not match the target expression, a pop-up window appeared prompting the participant to continue entering the expression. Otherwise the participant was given on-screen instructions on how to continue.

After the practice trials the participants entered seven expressions without the aid of the experimenter. First, participants entered expressions one to three (see Table 1) in either the Align-only or Render-only condition. Next, participants re-entered the same expressions with the same ordering in the opposite condition. A counter-balanced design was adopted in order to minimize practice effects and increase internal validity [8]: order of condition and expression presentation for the first two conditions and condition order for the practice trials were randomized within participants. For the seventh and final trial the participants entered expression four (see Table 1) in the third condition (FFES with Render and Align). Finally, participants completed the questionnaire.

5.2 Results

Timed Expression Entry Results

The means for expression entry times, the number of Stop presses and number of Align and Render presses are shown in Tables 2, 3, and 4. Three 2 $\times$ 3 within subjects ANOVA tests were used to examine the contribution of condition and equation to entry times, the number of times Stop was pressed, and the number of times Align


Freehand Formula Entry System - /home/zanibbi/morffes/RichardTests/last.ffes*
File Edit Mode Operations Help Display
MODES
OPERATIONS
Draw
Group Strokes
Move
Delete
Label
Undo
Redo
RENDER
ALIGN
STOP
Typeset Expression
Target Expression
Draw Mode

$$\int_{-\infty}^{+\infty} \frac{(2^x+4x)}{-z} dx$$

$$\int_{-\infty}^{+\infty} \frac{(2^x+4x)}{-z} dx$$

Figure 4: A screen shot of the experiment in progress.

and Render were pressed in the first two conditions. For all three ANOVA there was a main effect for expression independent of condition (entry time: ($\underline{F}$=204.26, $\underline{p} < .05$), Stop presses: ($\underline{F}$=8.04, $\underline{p} < .05$), Align and Render presses: ($\underline{F}$ = 19.25, $\underline{p} < .05$)). Independent of expression there was a significant difference between the two conditions (Render and Align) in the number of Stop presses ($\underline{F}$=13.14, p $< .05$). However, there was no significant difference between conditions for entry time or the number of times that Align and Render were pressed ($\underline{p} >$ .05).

Paired samples t-tests between the Render and Align conditions for each equation were not significant for entry time or the number of Render and Align presses ($\underline{p} >$ .05). A t-test showed a significant difference between the number of times Render and Align were pressed in the final condition ($\underline{t}_{26}$=5.24, $\underline{p} < .05$). The t-tests for number of Stop presses were significant between conditions for expressions two and three (($\underline{t}_{26}$=2.32, $\underline{p} < .05$) and ($\underline{t}_{26}$=2.78, $\underline{p} < .05$) respectively).

Questionnaire Results

When asked if the Align operation was useful, 16 (59.3%) participants responded Yes, while 11 (40.7%) responded No. All participants (100%) indicated that Render was useful. Twenty-three participants (85.2%) reported that they felt that using Render was faster than using Align, three participants (11.1%) felt neither was faster, and one participant (3.7 %) did not respond. When asked which of Align or Render allowed easier correction of interpretation errors, 22 (81.5%) replied Render, four participants (14.8%) replied Align, and one participant (3.7%) replied Neither.

Fifteen participants (55.6%) reported that FFES with Render and Align allowed the simplest correction of interpretation errors overall, while 12 participants (44.4%) reported finding FFES with Render easiest to correct interpretation errors with. When asked which version of FFES they enjoyed using most, 16 participants (59.3%) replied FFES with Render and Align, and 11 participants (40.7%) replied FFES with Render only.

The FFES version participants enjoyed using most was correlated with entry time for expression one in the Align condition ($\underline{r}$=-.58, $\underline{p}$<.05), the number of Align presses for expressions one and two (($\underline{r}$=-.45, $\underline{p}$<.05) and ($\underline{r}$=-.41, $\underline{p}$<.05), respectively) and the number of Align presses in the final condition ($\underline{r}$=.52, $\underline{p} < .05$).

Twenty-four participants (88.9%) were interested in using a system similar to FFES again, while three (11.1%) were not.

5.3 Discussion

Our hypothesis that Render is more effective in communicating the system's interpretation was supported by the

Expression	1	2	3	4
Render	49.42 (SD= 23.08)	159.62 (SD=58.67)	199.42 (SD= 73.24)	
Align	51.58 (SD= 27.92)	166.58 (SD= 85.15)	226.42 (SD= 74.86)	
Both				172.5 (SD= 78.37)

Table 2: Mean Entry Time in Seconds

Expression	1	2	3	4
Render	1.0 (SD= 0.0)	1.37 (SD= 0.69)	1.19 (SD= 0.49)	
Align	1.35 (SD= 1.13)	2.42 (SD= 2.37)	2.15 (SD= 1.64)	
Both				1.31 (SD= 0.88)

Table 3: Mean Number of Stop Presses

fact that users pressed Stop significantly more often under the Align-only condition. Pressing stop more than once per formula indicates that the user thought that the system was interpreting their formula correctly when it was not. This may result from two factors. First, Align does not provide unambiguous feedback about the symbol labels. Second, some participants found the symbol layout of the Align operation confusing.

Our hypothesis that time to enter a correctly interpreted formula would be greater for Align was not supported. Despite slower feedback and incomplete information, the overall process was just as fast using Align as with Render.

When given the option, participants used Render more often than Align. However, participants that enjoyed using FFES with Render and Align the most used the Align operation more in the final condition than those that preferred FFES with Render only. The participants who reported preferring the version of FFES with Render and Align also appear to have been more proficient with the Align operation, evidenced by correlations with faster entry time for expression one in the Align condition, and fewer Align presses for expressions one and two. It may be that some individuals are simply more comfortable than others with the type of feedback that the style-preserving morph provides.

Finally, participants were poor at determining under which of Render or Align they entered expressions more quickly or corrected interpretation errors more easily. There were no correlations between participants' responses on these matters and any other data analyzed.

5.4 Usability Feedback

Participants that found Align useful reported that this was because it provided feedback on the relative positions of symbols, formatted their input in a pleasing way, and in a few cases helped them find an interpretation error in the final condition.

a. Original Input b. After style-preserving morph

Figure 5: Adjacent symbols can appear to have a subscript relation after using Align.

Participants who did not find Align useful stated that this was because the Align results were often confusing. For example consider Figure 5, where after using Align the positioning of the adjacent '$=$' relative to the 'E' appears subscripted. Some participants also strongly disliked that their input changed at all, or the way that the formatting algorithm resized their symbols (e.g. if a participant wrote short, wide symbols, after pressing Align the symbols would shrink substantially). Another common complaint was that Align did not provide symbol recognition feedback the way that Render did.

Participants reported that Render was useful because it showed symbol recognition results, was easier to detect errors with than Align, and the resulting bitmap was more familiar, aesthetically pleasing and easier to compare to the target than the style-preserving morph result.

General feedback obtained regarding FFES was in line with an earlier published study on the system [13].

6 Conclusion

This paper introduced the notion of style-preserving morphs for generating unobtrusive feedback from interpretive interfaces. Style-preserving morphs employ gradual changes to preserve the user's mental map of the input, and use affine transformations of the input strokes to preserve the user's writing style.

Our user study showed that despite being both slower and less precise in its feedback than a conventional type-

Expression	1	2	3	4
Render	1.35 (SD=0.49)	3.04 (SD= 2.37)	3.12 (SD= 2.22)	2.81 (SD= 2.06)
Align	1.54 (SD= 1.24)	2.89 (SD= 2.20)	3.23 (SD= 2.10)	0.962 (SD= 1.15)

Table 4: Mean Number of Align and Render Presses

set rendering, users are able to get the job of equation entry done just as quickly using only a style-preserving morph. We believe that the style-preserving morph makes it easier for users to correspond their input to the interpreted output, and that this benefit is compensating for the method's other weaknesses. However, more experimentation is needed to determine the cognitive effects of each of the feedback methods.

While none of the users saw Align as supplanting the Render operation, most felt that it was a useful addition to the interface. Further development of the Align operation will make it even more useful. The appearance of symbols after morphing can be improved by taking into account areas rather than simply heights and widths, and confusing vertical alignments of adjacent baseline symbols such as seen in Figure 5 may be prevented with some simple modifications of Step 2 in the formatting algorithm.

Feedback by its very nature always involves some disruption. Despite our attempts to make the style-preserving morphs as unobtrusive as possible, some users disliked the fact that their input was tampered with at all. It seems likely that tolerance for disturbance of one's writing style or mental map varies widely among individuals. If so, it may help to allow the user to adjust the extent to which the system is allowed to modify the user's drawing area.

Acknowledgements

The authors wish to thank Dorothea Blostein for helping make this collaboration possible, and Jim Cordy for providing the TXL compiler. The authors also wish to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada and the National Science Foundation (USA), under Career Award CCR9876332.

References

[1] James Arvo. Computer aided serendipity: The role of autonomous assistants in problem solving. In *Proceedings of Graphics Interface '99*, pages 183–192, Kingston, Ontario, June 1999.

[2] James Arvo and Kevin Novins. Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes. In *ACM Symposium on User Interface Software Technology*, Stanford, California, November 2000.

[3] James Arvo and Kevin Novins. Smart text: A synthesis of recognition and morphing. In *AAAI Spring Symposium on Smart Graphics*, pages 140–147, Stanford, California, March 2000.

[4] J.R. Cordy, C.D. Halpern, and E. Promislow. Txl: A rapid prototyping system for programming language dialects. *Computer Languages*, 16(1):97–107, Jan 1991.

[5] Julie Dorsey and Leonard McMillan. Computer graphics and architecture: State of the art and outlook for the future. *Computer Graphics*, 32(1):45–48, February 1998.

[6] Takeo Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka. Interactive beautification: A technique for rapid geometric design. In *ACM Symposium on User Interface Software and Technology*, pages 105–114, 1997.

[7] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. In *Computer Graphics Proceedings*, Annual Conference Series, pages 409–416, August 1999.

[8] G. Keppel. *Design and Analysis: A Researcher's Handbook*. Prentice Hall, Englewood Cliffs, NJ, second edition, 1982.

[9] K. A. Lyons, H. Meijer, and D. Rappaport. Algorithms for cluster busting in anchored graph drawing. *Journal of Graph Algorithms and Applications*, 2(1):1–24, 1998.

[10] I. S. MacKenzie and S. X. Zhang. The immediate usability of graffiti. In *Proceedings of Graphics Interface '97*, pages 129–137, 1997.

[11] L. Markosian, M. A. Kowalsi, S. J. Trychin, and L. D. Bourdev. Real-time nonphotorealistic rendering. In *Computer Grahpics Proceedings*, Annual Conference Series, pages 415–420, 1997.

[12] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages and Computing*, 6:183–210, June 1995.

[13] Steve Smithies. Freehand formula entry system. Master's thesis, Department of Computer Science, University of Otago, Dunedin, New Zealand, May 1999.

[14] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Proceedings of Graphics Interface '99*, pages 84–91, Kingston, Ontario, June 1999.

[15] Osama Tolba, Julie Dorsey, and Leonard McMillan. Sketching with projective 2D strokes. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology*, pages 149–157, Asheville, North Carolina, November 1999. ACM Press.

[16] Richard Zanibbi. Baseline structure analysis of handwritten mathematics notation. Submitted for publication.

[17] Richard Zanibbi. Recognizing mathematical expressions using tree transformation. Submitted for publication.

[18] Richard Zanibbi. Recognition of mathematics notation via computer using baseline structure. Technical Report ISBN-0836-0227-2000-439, Department of Computer Science, Queen's University, Kingston, Ontario, Canada, August 2000.

Appendix A: Questionnaire

1. What is your gender?
2. What is your age?
3. What is your occupation?
   1. Undergraduate Student, Discipline:
   2. Graduate Student, Discipline:
   3. Professor, Discipline:
   4. Other (please specify):
4. Which one of the first two versions of FFES that you used allowed you to enter expressions most quickly (check one)?
   1. FFES with Align Symbols
   2. FFES with Render
   3. Neither
5. Which one of the first two versions of FFES that you used allowed you to correct interpretation errors most easily (check one)?
   1. FFES with Align Symbols
   2. FFES with Render
   3. Neither
6. Which one of the three versions of FFES that you used did you enjoy using most (check one)?
   1. FFES with Align Symbols
   2. FFES with Render
   3. FFES with both Align Symbols and Render
7. Would you be interested in using a system similar to the Freehand Formula Entry System in the future?
   1. Yes 2. No
8. In which of the three versions of FFES used in the experiment was it easiest to correct interpretation errors (check one)?
   1. FFES with Align Symbols
   2. FFES with Render
   3. FFES with both Align Symbols and Render
9. Overall, was the Align Symbols operation useful?
   1. Yes 2. No
   Please provide a brief reason for your answer.
10. Overall, was the Render operation useful?
   1. Yes 2. No
   Please provide a brief reason for your answer.
11. Please explain briefly what in your opinion was the strongest part of the program.
12. Please explain briefly what in your opinion was the weakest part of the program.
13. Please provide any additional comments you have about FFES.