<math>ml INTERNATIONAL CONFERENCE 2002 HOSTED BY WOLFRAM RESEARCH

> Call for Papers

> General Information

> Registration

> Accommodations

> Travel

> Tutorials

> Presentations

> Schedule

An Interactive Mathematical Handwriting Recognizer for the Pocket PC

Bo Wan and Stephen Watt
The University of Western Ontario

Abstract

Handwriting is a highly desirable method to input mathematics for computer applications, including pen-based electronic devices such as tablet PCs and PDAs. This paper proposes a method for mathematical handwriting recognition suitable for environments with strong resource limitations, such as PDAs, where the usual approaches to recognizing handwritten math are not feasible. We describe an experimental on-line mathematical recognizer for a specific PDA, the pocket PC. This recognizer handles handwritten mathematical expressions dynamically and generates the corresponding presentation MathML as output.

Our implementation consists of a recognizer for individual handwritten mathematical symbols, a structural analyzer for recognizing the relationship between recognized symbols of an expression, and a procedure for generating corresponding presentation MathML code.

Keywords

handwriting recognition, mathematical expression recognition, presentation MathML, Pocket PC

1. INTRODUCTION

Since PDAs are too small to have conventional keyboards, there has been considerable attention devoted to the use of pen-based input in this setting. In this setting, and in the future on Tablet PCs, we naturally see a host of well-developed handwriting recognition applications. A special-purpose recognizer is necessary to be able to work with mathematics, as the two-dimensional layout and use of a large number of symbols of multiple sizes --from small to very large-- makes mathematical handwriting just as similar to drawing as it is to normal handwriting.

The existing handwriting recognizers for PDAs are unsuitable for two reasons: 1) they are not able to recognize most of the mathematical symbols, and, more importantly, 2) they assume that symbols are always written linearly in order, no matter how the user writes the symbols. This is not the case, however, for mathematics where the symbols are arranged in a two-dimensional structure and the spatial relationship between the symbols has embedded mathematical meaning. Recognizing handwritten math includes recognizing both individual symbols and the two-dimensional structure.

The existing handwriting mathematical recognizers on desktop or laptop computers may not be suitable for PDAs either. Most of these recognizers perform the structural analysis by means of formal languages, which are require resources of a magnitude not available in a PDA. Secondly, these methods tend to be off-line (analyzing whole formulas at once) and in a PDA it is most appropriate to on-line (incremental) methods.

In this paper we describe a handwriting recognition method suitable for use in a resource-constrained environment, and prototype implementation for a specific PDA, the Pocket PC. Our goal has been to find an effective way to recognize handwritten mathematics on PDAs, with a focus on the recognition of spatial relationships between symbols. This implementation consists of three components:

1. a symbol recognizer to recognize individual handwritten mathematical symbols,
2. a structural analyzer to interpret and maintain the relationship between recognized symbols of an expression, and
3. a procedure to generate presentation MathML code.

This is an on-line system: every time a new symbol is finished, it will be recognized immediately. The structural analyzer then analyzes the spatial relationship between the newly recognized symbol and other symbols of the expression.

Our system is currently based on relatively simple recognition modules. Nevertheless, it functions quite well for a small set of mathematical symbols and relatively simple mathematical expressions such as polynomial equations, fractions, and trigonometric functions.

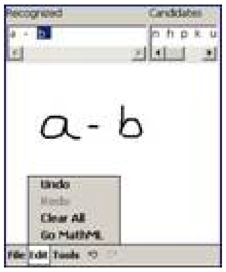
2. USER INTERFACE

Our implementation is based on Microsoft Embedded Visual C++, which is designed to operate in the environment of Windows CE (the dominant operating system for Pocket PCs).

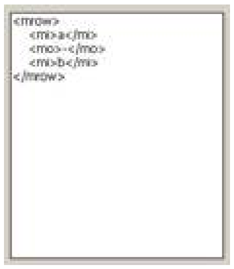
The user interface is dominated by a window where the user writes mathematical expressions (the input window). This window is used as a drawing canvas in order to capture the spatial information of the expression. Above the input window there are two panels: the left panel displays the identity of the recognized symbols, with the current symbol highlighted; the right panel shows a list of other candidates for the current symbol. This usually allows a mis-recognized symbol to be corrected with a single pen-tap on the correct candidate.

At any time the user can view the presentation MathML for the expression by selecting the menu entry "Edit/Go MathML".

Figure 1 shows screen shots of the program.



(a)



```
<mrow>
   <mi>a</mi>
   <mo>-</mo>
   <mi>b</mi>
</mrow>
```

(b)

Figure 1 User interface of the mathematical recognizer.
(a) Input area and panels for recognition result and candidates.
(b) A pop up window displays the corresponding MathML code

3. SYMBOL RECOGNITION

Handwriting recognition has been studied for nearly forty years and there are great many proposed approaches. The problem is quite complex, and even now there is no single approach that solves it both efficiently and completely in all settings.

While the principal contribution of this paper is in the area of structural analysis, we found that it was necessary to provide an implementation of handwritten symbol recognition suitable for a mathematical setting. In our symbol recognizer, the user can write a symbol either as a single stroke or with multiple strokes. The recognition algorithm we have used is elastic matching, which is model based. Once recognized, we represent the mathematical symbols as Unicode characters.

3.1 Stroke Grouping

In our system, symbol recognition is performed at the symbol level. However, a symbol is input at the stroke level, and it is not trivial to segment strokes into symbols. We handle this problem by examining the time delay and spatial distance between a new stroke and the previous stroke. We assume that the user always finishes the strokes of one symbol before moving on to the next one. If either the time delay or the spatial distance exceeds a corresponding threshold, we make the new stroke the first stroke of a new symbol. Otherwise the strokes are considered as belonging to the same symbol. Both of the thresholds can be tuned to a particular user's writing habits.

3.2 Elastic Matching

There are many approaches for on-line symbol recognition, but not all of these are suitable for use on a PDA. The main constraint is imposed by hardware limitations, such as processor speed and storage capacity. For example, neural networks are usually too large in size to fit in present pocket PCs. Some approaches, such as the system of Chan and Yeung [1], use formal languages in the recognition. If used on PDAs, the recognition speed may be very slow due to the parser's complexity.

We have yet not found any algorithm that is ideal for PDA devices (for both recognition rate and performance). We therefore use the best compromise algorithm we have encountered, elastic matching. This method is commonly used in on-line handwriting recognition [2]. It is robust and handles writing errors quite well. As a model-based technique, it calculates the distance between the unknown symbol and models using a dynamic programming algorithm. It then selects the model which minimizes the total distance to the unknown symbol.

When a vocabulary of a large number of symbols is in use, this approach will be too slow, and it will be necessary to divide the symbols into classes based on some basic feature analysis. Since our main focus is on the structural analysis step, at this time we have not put too much effort on the symbol recognizer, instead we have implemented it in a simple way. More details of the algorithm can be found in [2].

4. STRUCTURAL ANALYSIS

The ability of a recognizer to recognize mathematical expressions depends on the understanding of the two-dimensional structure. This is not trivial due to the nature of mathematics its explicit and implicit rules.

4.1 Symbols and Operators

In mathematical expressions there are different types of symbols, each type has its own grouping rules. We group them as follows:

- **Basic symbols** are usually considered as separate units. However, digits can sometimes be grouped together as one unit representing a number (*e.g.* 123), and Roman letters can sometimes be grouped together as one unit representing a function name (*e.g.* sin).

- **Binding symbols** such as Σ, $\int$, , and fraction bar dominate the sub-expressions around them and group those sub-expressions as individual units.
- Pairs of **fence symbols** group the sub-expression enclosed in them as one unit.
- There are **context sensitive** cases in which the same combination of symbols has different semantics. For example, in the expressions $\int xdx$ and $dx+cy$, dx has different meanings.

We can interpret mathematical expressions as also having implicit operators, encoding the spatial relationship between symbols. For example the superscript relationship between *a* and *2* in a^2 can represent a power of *a*. Correctly recognizing the spatial relationship between symbols and subexpressions is critical for recognizing implicit operators. This is not an easy task, however, as noted in [3], because "mathematical notation" is not formally defined, and is only semi-standardized, allowing many variations. One related problem is that there is no clear separation of positions between horizontal adjacency and superscript or subscript, and it will be hard sometimes to determine the relationship between symbols. For example, in the following expressions: (1) a^2, (2) a^2, (3) a^2, (4) a^2, the relationships between *a*and *2* changed from row to superscript, however the relationship in expression (2) and (3) are hard to tell.

4.2 Earlier Approaches to Structural Analysis

Existing mathematical expression recognition systems use a variety of approaches to analyze the two-dimensional arrangement of symbols and subexpressions. Reviews of this subject can be found in the survey papers of Blostein and Grbavec [3] and Chan and Yeung [4]. Following Blostein and Grbavec, these methods can be roughly classified into four groups: syntactic methods, projection-profile cutting, graph-rewriting, and procedurally-coded rules. Of these the projection-profile cutting approaches are mainly used for off-line systems. All the syntactic methods and graph-rewriting methods have used some syntactic grammar or graph grammar in the analysis. As a result these approaches are computationally expensive and may not be suitable for a resource-constrained environment, such as a pocket PC. Additionally, in our system the structural analysis must be performed dynamically. Every time a symbol is recognized, its relationship with other symbols must be incrementally analyzed, and this may imply that the expression tree is changed dynamically. Grammars cannot readily satisfy this requirement. In our system we used a procedural code approach, which is detailed in the following sections.

4.3 A Geometric Approach

Our approach determines the relationship between symbols and subexpressions based on bounding box information. A bounding box represents the spatial location and extent of a sub-expression. There are several useful measures of distance and direction between bounding boxes: Distance can be measured as the separation between nearest edges of the boxes, or the max norm or euclidean distance between the geometric or typographic centers or between the origins of the bounding boxes. (We use the term "typographic center" to refer to the natural center point of the typographic baseline of an expression.) Likewise, direction may be calculated either between geometric or typographic centers of the bounding boxes.

We use the geometric relationship between adjacent boxes to determine implicit mathematical operations, and form larger bounding boxes around the recognized sub-expressions. This process is repeated until a tree structure of nested bounding boxes is completed. In many ways, this is similar to operator precedence parsing. In practice it is necessary to know a writing direction, and unless otherwise specified, we assume that symbols are written from left to right horizontally.

4.4 Bounding Boxes and Direction Calculation

For a symbol the bounding box is considered as the smallest rectangle that encloses it. In general we assume that the typographic center of a symbol's bounding box is the midpoint of the bottom of the bounding box. For certain symbols (*e.g.* "b" or "p") this must be adjusted to take ascenders and descenders into account. Due to the fact that we do not have a symbol structure extraction module at this time, we assume the ascender or descender to be 40% of the symbol, when applicable.

For an expression, the bounding box is calculated from the convex hull of all the sub-expressions. To ensure that the baseline of the dominant sub-expression becomes the baseline of the whole expression, we use rules based on the mathematical operation implied by the spatial relation. For example, if two boxes are found to be in a superscripting relative position, the bounding box of the whole expression will have its baseline set as the baseline of the base (non-superscript) sub-expression. In general, each sub-expression of the entire expression will use its own method to calculate its bounding box and baseline.

Currently, in the structural analysis phase we consider only five relationships: *row*, *above*, *under*, *superscript*, and *subscript*. We also require that the handwritten symbols be well separated (not touching). The direction between two bounding boxes is determined as follows: if the *X*-projection of one box is mostly contained in that of the other, then the relationship is considered as *above* or *under*, depending on their relative position. Otherwise, if the difference between the *Y*-coordinates of their centroids is less than a threshold then it is considered as *row* relationship. In other cases the relationship is determined according to the angle formed by the bounding boxes' centroids.

4.5 Group Determination and Tree Updates

We realize that the key to the structural analysis is to correctly identify the appropriate grouping unit for each symbol and sub-expression. This is non-trivial, as each type of mathematical symbol has different grouping rules, and the rules can be short range or long range.

For a newly recognized symbol, we start by searching the expression tree for the symbol node that closest in distance to the new symbol. This node is called the Nearest Neighbor (NN) node, and it is the start point for determining the appropriate grouping.

We first check whether the relationship between the new node and the NN node matches any of the special grouping rules. For example, if row relationship is formed and both symbols are digits then they can be considered as one unit. As another example, if the NN node is an opening parenthesis to the left in a row, then they form a grouping unit with the NN node being the parent node of the new node.

In cases where the relationship between the new node and the NN node matches none of the general grouping rules, we perform the further analysis. These may involve non-local grouping. For example, in the expression $a^{b+c}d$,d is in a grouping unit with the sub-expression a^{b+c} instead of c, although c is the closest symbol to d

in distance. Our strategy is to check for any possible long range grouping unit for the new node. only when the possibility of long range grouping has been excluded do we check for short range groupings.

Our tests are performed in the order: *row*, *above* and *under*, *superscript* and *subscript*. The rules involved are complex and we do not describe all the details. The basic idea is to begin with the NN node and follow up the path to the root of the expression tree. At each level we check for the possibility of the sub-expression represented by the node to form a grouping unit with the new node. Then we choose the most appropriate candidate based on the context. For example, in the *row* direction check we check all the ancestor nodes of the NN node in the expression tree to see whether any of those is in a row relationship with the new node. If the exceptions are not considered, the top level candidate is determined as the one that forms a *row* grouping unit with the new node.

Once the relationship has been determined, the new node is attached to the expression tree. This will often require the rearrangement of the expression tree prior to inserting the new node. These rearrangements may involve splitting nodes, merging nodes, deleting some nodes, or moving some nodes around the expression tree.

In our approach the expression tree is maintained in such a way that it is straightforward to generate Presentation MathML on demand. At present, this is done only when explicitly requested, but it would be a simple matter to maintain a MathML tree and perform the editing operations directly upon it using DOM operations.

5. CONCLUSIONS AND FUTURE WORK

We have presented a geometric method for on-line mathematical expression recognition suitable for use in an environment with limited resources, and an embodiment in mathematical handwriting recognizer for the pocket PC.

The current implementation for the PocketPC is able to recognize simple polynomial expressions, fractions, and trigonometric functions. Nested structures can also be recognized properly. Although the implementation is rudimentary, it provides sufficient evidence that the geometric approach can work reasonably well in this context.

It remains to develop patterns for enclosing expressions, such as square roots and matrices. It is also necessary to experiment with ways to handle the input of large expressions on a small input area. Ultimately, it would be interesting to experiment with this sort of handwriting recognition exchanging MathML data with a system such as Maple through a DOM interface on a Pocket PC.

REFERENCES

[1] K.F. Chan and D.Y. Yeung. *Elastic Structural Matching for On-line Handwritten Alphanumeric Character Recognition.* Technical Report CS98-07, Department of Computer Science, Hong Kong University of Science and Technology, March 1998.

[2] P. Scattolin. *Recognition of Handwritten Numerals Using Elastic Matching*. Master thesis, Department of Computer Science, Concordia University, Montreal, Quebec, 1995.

[3] D. Blostein and A. Grabvec. *Recognition of Mathematical Notation.* in H. Bunke and P.S.P. Wang (editors) Handbook of Character Recognition and Document Image Analysis, pp. 557-582, World Scientific, Singapore, 1996.

[4] K.F. Chan and D.Y. Yeung. *Mathematical Expression Recognition: A Survey.* International Journal on Document Analysis and Recognition, 3(1): 3-15, August 2000.