Reprinted from INTERCHI '93 Conference Proceedings, Amsterdam, April 1993, pp. 80-87

Touch-Typing With a Stylus

David Goldberg and Cate Richardson

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Rd.
Palo Alto, CA 94304
goldberg@parc.xerox.com
(415)-812-4423

ABSTRACT

One of the attractive features of keyboards is that they support novice as well as expert users. Novice users enter text using "hunt-and-peck," experts use touch-typing. Although it takes time to learn touch-typing, there is a large payoff in faster operation.

In contrast to keyboards, pen-based computers have only a novice mode for text entry in which users print text to a character recognizer. An electronic pen (or stylus) would be more attractive as an input device if it supported expert users with some analogue of touch-typing.

We present the design and preliminary analysis of an approach to stylus touch-typing using an alphabet of *unistrokes,* which are letters specially designed to be used with a stylus. Unistrokes have the following advantages over ordinary printing: they are faster to write, less prone to recognition error, and can be entered in an "eyes-free" manner that requires very little screen real estate.

KEYWORDS

Stylus, electronic pen, handwriting, printing, recognition, text entry, pen-based computing, shorthand.

INTRODUCTION

Keyboards are a vital part of today's computers. Although keyboards are somewhat bulky, they are well suited to PCs (even portable laptops) and workstations. In the future of Ubiquitous Computing [17], pocket-sized and wall-sized computers will be common. A keyboard is not very suitable for these sizes of computers. Although some manufacturers have put tiny keyboards on hand-held computers, such small keyboards are hard to operate, and impossible to use for high speed touch-typing. Similarly, keyboards do not work well for large wall-sized displays [3], because a keyboard is fixed and can't be reached from all parts of the display.

Thus many manufacturers are providing electronic pens or styli (we use the two terms interchangeably) as the primary input device for computers. A stylus is attractive because it works very well over the entire range of sizes. However, it is not very convenient for text entry. The state of the art is to print characters, with boxed entry recommended to improve accuracy [1]. This is slow and error prone [10]. Although it is true that computer interfaces use text input more than necessary (compare the Macintosh or MS-Windows with the older DOS or UNIX shell), considering how much of our daily lives involves reading and writing, it seems likely that interacting with computers will involve a significant amount of text entry for a long time to come. This suggests that a major impediment to the widespread use of styli is the problem of finding a convenient way to enter text.

Some manufacturers of pen computers suggest that the solution to this problem is to use uninterpreted handwritten text. Although this works well for scribbling a note in a personal calendar, uninterpreted text is not suitable for composing even a short memo if it needs to be filed in a form that can be later searched. Thus the problem this paper will address is: what is a convenient way to enter *interpreted* text?

There is an analogy between keyboards and styli. Keyboards can be used with no training: the letters can be tapped out one-by-one using hunt-and-peck. This is similar to what is currently done with styli. No new training is required, and letters are printed one-by-one. However, unlike styli, keyboards have a "growth path." With practice, hunt-and-peck with two fingers can become faster than handwriting. If even higher speeds are desired, then keyboard users can learn touch-typing. Touch-typing not only achieves high speeds, it also enables "eyes-free" operation, that is, the ability to type without having to look at your hands. This suggests that the solution to the problem of stylus text entry requires developing an analogue of touch-typing.

UNISTROKES AND HEADS-UP WRITING

Our approach to developing touch-typing for a stylus is based on introducing a special alphabet. Like touch-typing for keyboards, this is a system that has to be learned.

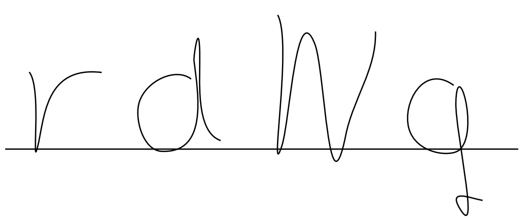
The traditional secretarial shorthand systems of Gregg and Pitman are one possible candidate for stylus touch-typing. Although these shorthand systems achieve very high speeds, they are at least as difficult to recognize as cursive handwriting, which is to say, using them for interpreted input does not appear practical at the present time [6] [8].

Shorthand systems can be classified as orthographic or phonetic. Orthographic systems use conventional spelling, and have one symbol for each letter of the ordinary Roman alphabet (this paper only considers text entry in English). Phonetic systems spell phonetically, and in the case of Gregg and Pitman, use a special phonetic alphabet. Phonetic systems are harder to learn, but can achieve very high speeds. This suggests that designing a stylus alphabet involves a trade-off between speed and ease-of-learning. Phonetic systems offer the fastest speed and are the hardest to learn, while orthographic systems are easiest to learn, but give a smaller speedup.

The system described in this paper is designed for ease of use, and thus is orthographic. Although its speed advantage over ordinary printing is modest, it has two other important advantages described in the next two sections: it is more accurate and it supports *heads-up writing*. If even higher speeds are desired, the system of this paper can be combined with techniques from existing shorthand systems that use the ordinary alphabet, such as Speedwriting [13].

Sloppiness Space

Ordinary Roman printed characters are not very robust in the face of rapid (hence sloppy) writing, since there are many pairs of letters that blur together when written quickly. For example, in the following figure, is the first letter an 'r'



or a 'v'? The second an 'a' or a 'd'? The third an 'N' or a 'W'? The fourth a 'g' or a 'q'? Although in any alphabet it is possible to draw symbols halfway between two valid letters, the ambiguities in the figure occur quite commonly in practice.

We have developed a system using *unistrokes*, which are designed somewhat like error correcting codes: they are well separated in *sloppiness* space. To explain this term, imagine that each unistroke letter is described by *d* features, and so can be thought of as a point in a *d*-dimensional space. Sloppiness space results when the features are chosen so that the changes caused by writing a letter more quickly (hence more sloppily) correspond to small changes in a letter's position in the *d*-dimensional space. Thus unistrokes that are well separated in sloppiness space can be robustly distinguished even when written sloppily. Sloppiness space is a useful concept even if the precise set of features defining it is not known.

Because unistrokes are designed to be well separated in sloppiness space, they have a higher accuracy rate than ordinary printed Roman letters, and thus the net speed of producing a correct document will be higher than might be suggested by raw input speeds.

Heads-up Writing

The second advantage of unistrokes over ordinary Roman letters arises because each unistroke is a single stroke (pen-down/pen-up motion), hence the name *unistroke*. There are no symbols like 'f' or 'H' that require multiple strokes. This is very important, because it enables heads-up writing, as will now be explained.

When writing with a pen or pencil, each successive letter must be in a different spot, for the simple reason that if one letter is written on top of the next, the result is a jumble that can't be easily read. However, when writing with a stylus, the computer sees each stroke as it is written, and writing a new letter on top of a previous one does not affect the information already recorded for the earlier stroke. There is a problem implementing this using the ordinary Roman alphabet due to letters consisting of multiple strokes. For example, writing 'T' followed by 'H' gives a total of five strokes, and is not obvious that these five strokes should be grouped into two strokes for the 'T', and then three strokes for the 'H'. This is not a problem with unistrokes, where each letter is a single stroke.

Thus unistrokes lend themselves to writing each letter on top of another, which we call heads-up writing. This has several advantages:

- *Little writing area is needed*. Heads-up writing is especially convenient on very small computers. A writing space need only be as large as the space needed for one letter.
- *Eyes-free operation.* Like touch-typing, heads-up writing does not require writers to look at their hands while writing. This is useful for transcribing text, for taking notes in a lecture, for writing in dim rooms, etc.
- *Easier on the wrist*. Heads-up writing does not require wrist movement when writing and is less fatiguing than ordinary handwriting. Although this may sound a bit far-fetched at first, it can be verified with an ordinary pen or pencil on a piece of paper. Try writing a sentence without moving your wrist, printing each letter on top of another. Most people find this experiment quite convincing.

The one drawback of heads-up writing is that the space separating words must somehow be indicated. Since space is much more frequent than any letter, we use a dot (that is, a tap of the pen) to indicate a space.

Discussion

One alternative to handprint recognition for text entry is to display a picture of a keyboard, and enter text by tapping on the displayed keys with a stylus. This is not as good as a real keyboard because there is no tactile feedback. Nonetheless, for some users it is faster than printing slowly enough to be reliably recognized.

The advantages of unistrokes over a displayed keyboard are that they require much less screen real estate, and that they support eyes-free operation. Unistrokes will probably be faster for most writers, because (with heads-up writing) precise positioning is not required for the start of each new stroke.

To summarize, we propose that "power" users who frequently use a stylus for text entry will benefit from learning a new alphabet consisting of single stroke letters called unistrokes. Although these letters can be written in a conventional manner, the full benefit of unistrokes comes from writing in a touch-print fashion, in which letters are written on top of one another.

THE DESIGN OF UNISTROKES

The three major criteria for designing unistrokes are, in order of importance: easy to learn, well separated in sloppiness space, fast to write.

The last goal is the simplest to implement. Straight strokes are faster to draw than curved or bent strokes. Thus unistrokes map frequent letters (e, t, a, i, r) to straight strokes.

Properly satisfying the second goal would require precise information about the features that describe sloppiness space. Lacking this information, our approach was to pick a set of strokes that appeared to be well separated, and then observe them being written in real use to see whether they were well separated in practice. The results are reported in the section on measurements.

The first goal was achieved by making many unistroke characters the same (or similar) to ordinary Roman characters. The unistroke alphabet has seven characters that are essentially identical to Roman letters and eight characters that are natural sub-strokes of Roman letters. Heuristics exist for most other characters.

Tricks

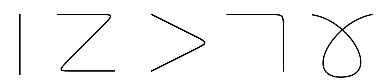
The design of unistrokes involves two "tricks." It is clear that the fewer characters that need to be encoded, the easier it is to design a set of unistrokes that are well separated. The character space can be divided in half by not having separate symbols for upper and lower letters. Thus the first trick is to use a button as a caps shift key, and to not have separate unistrokes for each case.

The simplest button to press is one on the side barrel of the stylus itself. For our current prototype systems, we use a stylus supplied by Scriptel Corporation which has a side button 1.5 inches from the tip of the pen. Although this is too high to be convenient for most users, upper case letters occur sufficiently infrequently so that this has not been a serious problem. For future work, we have designed our own stylus with a side button that is both easy to press and unlikely to be pressed by accident. For systems without a side button, an on-screen button can be used as a case toggle-switch.

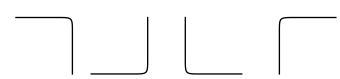
The second trick exploits a difference between ordinary pens and electronic pens. When looking at characters written with an ordinary pen, there is no simple way to distinguish a vertical stroke written from top to bottom from one written in the other direction, from bottom to top. But as seen by a computer, these strokes are totally different. Thus a vertical stroke can be used for two different unistroke characters. This is especially useful because strokes written in opposite directions are widely separated in sloppiness space.

The Unistroke Alphabet

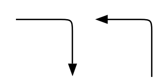
The alphabet of unistrokes is based on these five strokes.



Each stroke comes in four different orientations.



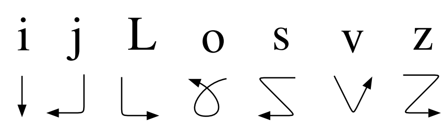
Furthermore, each stroke can be written in two directions.



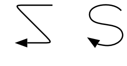
Since there are 5 basic shapes, 4 orientations, and 2 directions, this gives $5 \times 4 \times 2 = 40$ unistrokes, more than enough to encode 26 letters.

Assigning Unistrokes to Roman Letters

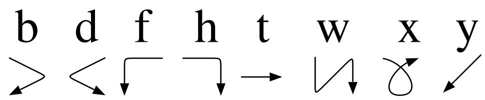
There are seven symbols that map directly to their ordinary representation in the Roman alphabet.



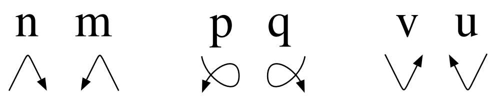
Although the unistrokes are shown with sharp corners, they can be drawn equally well with rounded corners (this is a small change in sloppiness space). Thus either of the two forms below could be written for 's'.



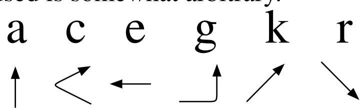
Next there are eight unistrokes that are subsets of the ordinary alphabetic characters. For example, the unistroke for 'b' represents the bowl of the 'b', etc. Actually, the unistroke for 'x' isn't a subset, but rather one way of writing an ordinary 'x' as a single stroke.



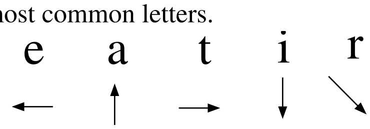
Then there are six letters that are best thought of as matching pairs. The letter 'v' has been seen before.



Finally, there are six "oddballs" with less natural mnemonics. For example, the unistroke for 'c' looks like the ordinary letter, but is written bottom-to-top. The unistroke for 'k' is one of the three straight line strokes of the 'k', but the one used is somewhat arbitrary.



Although the assignments of unistrokes was done primarily with an eye towards matching the ordinary alphabet as closely as possible, straight line strokes are assigned to five of the most common letters.



Discussion

It might seem that it would be simpler to develop an alphabet by directly stylizing each ordinary Roman printed letter. However, such a system will not be totally mnemonic. For example 'f' and 't' would have to be converted to single strokes, pairs close in sloppiness space (e.g. 'u', 'v') would have to be separated, and so on. By the time this is done, the resulting system would not be substantially easier to learn than unistrokes, and wouldn't benefit from the planning that exploited direction and mapped frequent letters to straight lines.

MEASUREMENTS

The full evaluation of a stylus text entry system should involve a preliminary design to get started, an evaluation of that design to pick out flaws, possibly several more design iterations, and then finally a head-to-head comparison with the best commercial pen user interface that uses ordinary printing. The development of unistrokes is still in the first design iteration. This section describes the results of evaluating the initial design described in the previous section.

We wanted to evaluate our system being used for a real task, rather than during some artificial experiment. Thus we built a mail sending program (Figure 1), and asked volunteers to send several messages per day using this program. The mail

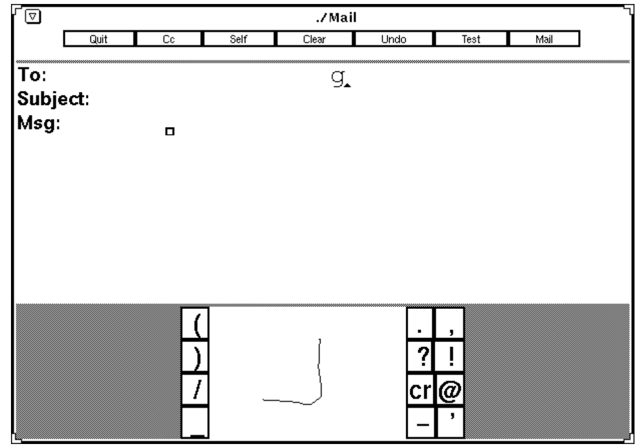


Figure 1 Stylus mail sending program

sender ran on a *scratchpad*, a unit containing a transparent tablet mounted on top of an LCD display, connected by a cable to the user's workstation [4].

Our measurements consisted of timings of strokes collected while running the mail sending program, recognition errors collected from a 'dictation' task, and informal impressions from users. The informal impressions were quite positive. About six people learned the unistroke alphabet by running a simple training program. All were able to correctly print unistrokes without looking them up on a reference sheet after about 10 minutes. Unfortunately, only three scratchpads were available, so only three users were able to use the mail sending program on a regular basis. They all seemed positive about using unistrokes as an alternative to ordinary Roman letters.

The Mail Sender and Stroke Timings

The mail sending program is written in Modula-3 [11], and is a front-end to the ordinary Unix mail program. It consists of three sections: a row of buttons at the top, a form in the middle, and a writing pad surrounded by buttons at the bottom. In the figure, the writing pad contains the unistroke for 'g'( ), which has been echoed in the 'To:' field. As characters are entered on the writing pad, they are echoed in the current field. The current field is changed by tapping with the stylus on the field labels. The mail is delivered by tapping on the 'Mail' button, which is the rightmost button on the top row.

The time spent during text entry has two parts, the time a stroke is actually written (from pen-down to pen-up), and the time between strokes (from pen-up to pen-down). The times spent writing strokes for one user as collected by the mail program are graphed in Figure 2. The value for each letter is the median of the collected times in milliseconds. As expected, dot (space) is fastest at 90 milliseconds, followed by straight-line unistrokes which cluster around 150 milliseconds. The fastest curved unistroke letter, 'j'( ), took 236 milliseconds to draw. Note that "retrograde"

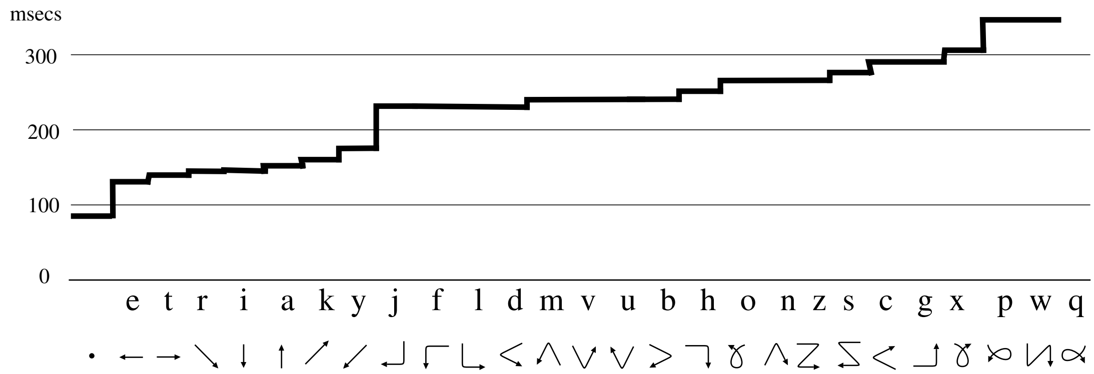


Figure 2 Median time in milliseconds spent drawing each unistroke

strokes such as ‘e’(←) which are written from right-to-left do not appear to be significantly slower to write than their more usual left-to-right counterparts. In fact ‘e’(←) is faster than ‘t’(→) and ‘m’(∧) is faster than ‘n’(∧). On the other hand, strokes written from bottom to top do seem to be somewhat slower (e.g. ‘c’ (<) and ‘g’(⅃)). To get some idea of the variation in times, Table 1 gives the statistics for space and the first ten letters from Figure 2.

Writing time for different strokes varied by about a factor of two (from 150 msecs to 300 msecs). The variation in the time spent between strokes was much larger. This is not surprising. After about ten minutes, users pass from the stage of looking up unistrokes on a reference card to the stage where they are memorized. At this point, inter-stroke timings depend on the speed of cognitive recall. As users gain more experience, writing unistrokes begins to become a motor skill. The variations in inter-stroke timings reflect this mixture of cognitive and motor skills. From the pairs of unistrokes that occurred at least five times for the user in Table 1, Table 2 lists the median of the inter-stroke times of the six fastest pairs and six slowest pairs. The median over all pairs was 158 milliseconds.

char	median	average	std dev	n
space	91	91	22	244
e	135	143	44	117
t	143	146	22	95
r	149	152	28	84
i	150	152	28	84
a	155	162	36	70
k	162	165	15	9
y	180	191	50	24
j	236	230	26	4
f	237	248	32	26
l	237	245	36	51

Table 1: Data for first ten unistrokes in Figure 2

pair	time	n	pair	time	n
mi	66	7	an	237	6
fi	82	6	ec	237	5
db	87	6	so	320	8
be	94	8	on	371	5
ne	94	7	ha	378	5
st	94	8	ou	577	9

Table 2: Fastest and slowest median elapsed times between pairs of unistrokes

One surprising thing in Table 2 is that the fastest times do not belong to pairs with the smallest travel time from pen-up to pen-down. It might be expected that combinations like ‘te’(→←), which consist of one stroke ending at the same point as the next stroke begins (assuming heads-up writing), would have the shortest inter-stroke times. There were in fact 9 ‘te’ pairs, with a median time of 151, very near the median of all letters. On the other hand, ‘mi’, the fastest pair, consists of an unmatched pair, with ‘m’(∧) ending low, and ‘i’(↓) starting high. This suggests that very little would be gained by changing the design so that the unistroke pairs of the most frequent digraphs (such as ‘th’, ‘in’, ‘er’) have little travel time.

Analysis of Timings

The primary purpose of collecting timings was to discover whether a different unistroke design would enable faster input speeds. Using the durations from Table 1, an inter-stroke time of 158 milliseconds, and letter frequencies obtained from outgoing mail logs, the average writing rate is 2.8 letters/second (throughout this section we ignore errors, and consider peak error-free writing speeds). If the same unistroke symbols were used, but the most frequent letter were mapped to the unistroke fastest to draw, and so on, the time would change to 3.0 letters/second. This is a gain of less than 10% in speed. Since this reorganization would make unistrokes harder to learn, and since speed gains much greater than 10% can almost certainly be achieved using techniques from shorthand (as in [13]), assigning unistrokes

pair	it	il	en	at	in
1 day	689	1102	419	369	524
1 week	157	221	237	299	303

Table 3: Median inter-stroke timings in msecs after one day, one week

entirely based on writing times does not appear to be a good design.

Shorthand textbooks suggest that the controlling factor in writing speed is not the time to write a stroke, but rather the pauses between strokes [5]. This is consistent with our data. For one of the writers measured, writing speed was 0.5 letters/sec the first day, and 1.0 letters/sec after a week. The first day, the average (weighted by frequency) time to write a unistroke was 344 msecs. After after one week, the average was 270 msecs, a modest gain. The median between-stroke pauses went from 1606 msecs to 654 msecs. Presumably certain pairs were being committed to motor memory. The fastest pairs after one week with at least three samples are given in Table 3, together with their timings after one day. From this table, it seems plausible that with further practice the median inter-stroke timing would drop to 300 msecs. Assuming the time to write strokes would remain the same, writing speed would approach 1.8 letters/sec.

A similar analysis for the user of Tables 1 and 2, who was writing at a rate of 2.8 letters/second, suggests that with further use the inter-stroke timing will drop to 100 msecs, giving a writing speed of 3.4 letters/sec. Not surprisingly, there is quite a variation in writing speeds from user to user. For comparison, the peak typing speeds of these two writers were 4.8 and 7.0 letters/sec, suggesting that writing speeds may be correlated with typing speeds.

As mentioned above, we used timings collected from everyday use of the mail program rather than perform artificial timing experiments. However, we did measure the user from Tables 1 and 2 repeatedly writing the sentence "touch typing with a stylus." The results agreed fairly well with the predictions above: the median inter-stroke time was 104 msecs, the writing rate was 3.2 letters/sec. For comparision, Van Cott and Kinkade [16] give values of 1.0–1.3 letters/sec for ordinary printing. However, this comparison needs to be viewed very cautiously because it doesn't control for different users, error rates, and so on.

Sloppiness space

In order to test how well unistrokes were separated in sloppiness space, each user wrote several dictated sentences, using a variant of the mail program. The sentence to be written was displayed, and no echoing was done until the complete sentence was written.

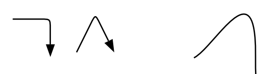
The dictation test turned up three recognition problems. The first has to do with dot detection. Although the difficulty of detecting dots has been recognized as a problem before [14], dots are used much more extensively in heads-up writing than in previous systems, since space is more common than any single character.

Dot detection is highly pen specific. For example, the program illustrated here also runs on the liveboard [3] which uses a different stylus technology from scratchpads, and requires a different dot detector. Even for a fixed stylus, we were unable to find a dot detector that worked universally well for all writers, although it was easy to find one that worked well for a given writer. Thus the first recognition problem had to be solved by introducing a writer-settable parameter for dot detection.

The second problem concerned pairs of unistrokes such as 'l' and 'i'

Some users would write strokes like the one on the right: half-way between an 'i' and 'l'. However, in practice these ambiguous symbols were always intended to be an 'l', so this second problem was solved by adjusting the recognition algorithm.

The third problem concerned pairs like 'h' and 'n'.



Some users wrote the stroke on the right, which is half way between 'h' and 'n'. One possible solution would be to change the unistroke for 'n' to be much sharper, like this:

and to interpret the ambiguous character above an 'h'. Further testing is required to validate this solution, and to check that this does not introduce new ambiguities. For example, if a stroke is drawn so narrowly that the two legs coincide, it will no longer be possible to distinguish between 'm'( ) and 'n'( ).

THE USER INTERFACE OF TEXT ENTRY

An earlier paper [4] argued that the user interface surrounding a text recognizer is at least as important as the recognizer itself. Unistrokes enable a different style of interface from current pen-based interfaces such as PenPoint.

In PenPoint, the default behavior for text entry uses a four step process. First a writing pad is created. There are two kinds of pads, popup and embedded. A popup pad has a single row of boxes, an embedded pad has several rows and takes up a substantial fraction of the screen. Then, text is entered by writing into boxes. After text entry, the next phase is entered by tapping on the 'OK' button, which causes a first attempt at recognition. During this phase, writing is interpreted immediately. The intention is that users will write corrections on top of misrecognized letters, although new text can also be entered. Finally, tapping on 'OK' a second time finishes the process, by dismissing the pad and causing the newly written text to be inserted into the document. In contrast, the user interface of the mail entry program has no modes. A small writing pad about the size of a single PenPoint letter box is always present, and at any time, text can be written on it, causing text to be inserted at the location of the caret.

Although the modeless interface of the mail sending program could be used with a conventional text recognition system such as that used in PenPoint, it is not a good match for the following reasons. First, keeping a pad permanently visible is only practical if it is small, and this implies heads-up writing, which works best with a unistroke alphabet. Second, because of the ambiguity of ordinary printing, a conventional text recognizer can do a better job if it sees a whole block of text before attempting to recognize, rather than recognizing on the fly. Third, when an error is made, a correction has to be written. With a boxed writing pad, the correction can be easily made right on top of the error. Things are not so simple in a modeless system, because the characters in the document are much more closely spaced than in the writing pad (by a factor of four in PenPoint). Once the incorrect letter has been inserted into the document, the close spacing of letters makes it difficult to write a new letter on top of the incorrectly recognized one. Thus a modeless system is not a good fit to a system when recognition errors are expected to be common.

The modeless interface of the mail program is quite similar to current keyboard interfaces, with the always visible writing pad playing the role of the always present keyboard. Heads-up writing with unistrokes is a much better match to this style of interface than ordinary printing. However, it is possible to provide a modeless interface to conventional recognizer. Casio sold a wristwatch and a palm sized "data bank" around 1984 that used a modeless heads-up writing interface. In these products, the user had to pause between each letter to enable the recognizer to group strokes into letters.

SUMMARY AND FUTURE WORK

It is our belief that the widespread use of styli would be greatly facilitated if there were an expert text entry system available, serving much the same function as touch-typing does for keyboards. This paper has explored basing such a system on a special alphabet.

A goal of our system was ease of use, so we chose an alphabet that is one-to-one with the ordinary Roman alphabet and is orthographic, that is, words are spelled in their usual fashion rather than phonetically. We call the symbols of our alphabet unistrokes, because each is a single stroke. Unistrokes are designed to be written unambiguously at high speed, and to exploit features present in a stylus but not in an ordinary pen, such as stroke direction and existence of a button to use as a caps shift-key. Although unistrokes can be written one next to another as in ordinary printing, to get the maximum advantage they should be touch-printed, that is, written one on top of another.

Early experience with our system has suggested that the unistroke design is within 10% of the maximum achievable in terms of speed, but can be improved in terms of accuracy. In our next version, we intend to modify the family of unistrokes including 'm'( ) and 'n'( ) to be better separated in sloppiness space.

All the users of the system reported that they assumed it would be hard to learn a new alphabet, but instead found that it was quite easy, because most unistroke symbols are the same as or very similar to their Roman letter counterparts. This suggests that it might be better to present unistrokes to users as a constrained version of the Roman alphabet, rather than as a new alphabet.

Future work can be divided into three areas. First, the current system needs to be refined and then tested head-to-head against the best commercial hand-print recognizer. One area for refinement is the separation of unistrokes in sloppiness space: the current design is quite *ad hoc* in this regard. It would be nice to bring knowledge about motor aspects of handwriting [7], [12], [15] to bear on this problem. Another area that needs to be refined is the method for entering punctuation and digits. In the mail sending program, punctuation is entered by pressing buttons surrounding the entry pad. Since period and comma are more frequent than letters such as 'q' and 'z', these two punctuation marks (at least) should be assigned unistrokes. There are at least four ways that digits could be entered: via buttons surrounding the entry pad, via buttons on a popup pad that appears in response to a special unistroke, using ten unistrokes from the basic set of 40 shapes, or by going into a special mode and using strokes resembling Arabic numerals. A special mode would be required in this last option to avoid ambiguity between (for example) '1' and '2' with the unistrokes for 'i'(↓) and 'z'( ).

A second area of future work is to explore other methods for stylus text entry. We have focused on printed characters. However, it may be possible to design a cursive alphabet that is easy to recognize. There are also approaches that don't use special alphabets at all. For example, the redundancy of English might be exploited as in the reactive keyboard [2].

Finally, there is the problem of developing a system that has the speed of touch typing with a keyboard. It is worth noting however, that products for speeding up typewritten input have not been commercially successful [9], so the importance of very high speed input may be overrated. We have estimated that unistrokes may be able to support a rate of as high as 3.4 characters/sec, whereas touch typists can achieve rates of 6-7 characters/sec. One way to close the gap would be to borrow ideas from shorthand. Given that shorthand systems using the Roman alphabet such as Speedwriting often perform better in practice than traditional Gregg and Pitman shorthand [18], the most promising avenue for further speedups may be to combine Speedwriting with the unistroke alphabet.

ACKNOWLEDGEMENTS

We owe a special thanks to Mark Stefik, who made the observation that writing letters on top of one another makes sense with a stylus, and to William Newman, who was an early advocate of unistrokes and not only had the idea of using a mail sending program for testing, but also wrote the first version of a mail sender. We also thank Tom Moran for his helpful comments on a draft of this paper, and Bill Buxton for informing us about the Casio data bank.

REFERENCES

1] Robert Carr and Dan Shafer. *The Power of PenPoint*, Addison-Wesley, 1991.

2] John J. Darragh, Ian H. Witten, and Mark L. James, “The reactive keyboard: A predictive typing aid” *IEEE Computer*, 41-49, Nov 1990.

3] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, *et, al*. “LiveBoard: A large interactive display supporting group meetings, presentations and remote collaborations,” *Proceedings of the Conference on Computer Human Interaction (CHI)*, 599-607, May 1992.

4] David Goldberg and Aaron Goodisman. “Stylus user interfaces for manipulating text.” *Proceedings of the Fourth Annual ACM Symposium on User Interface Software and Technology (UIST)* 127-135, Nov, 1991.

5] John R. Gregg, Louis A Leslie, and Charles E Zoubek, “Gregg Shorthand Dictionary”, McGraw-Hill, 1972.

6] Khaled Kamel and Ibrahim Iman, “A computerized transcription system for cursive shorthand writing”, *Proceedings of IEEE SouthEastcon*, Knoxville, Tennessee, 336-339, April 1988.

7] H. S. R. Kao, G. P. van Galen, R. Hoosain (eds) *Graphonomics, Contemporary Research in Handwriting*, North Holland, 1986.

8] C. G. Leedham and A. C. Downton, “On-line recognition of Pitman’s handwritten shorthand—an evaluation of potential,” *International Journal of Man-Machine Studies*, 375-393, 1986.

9] Bill Machrone, *PC Magazine*, pp 67-68, January 16, 1990.

10] Walter S. Mossberg, “Notepad PCs Struggle With One Small Task: Deciphering Writing.”, *Wall Street Journal*, Thursday July 23, 1992.

11] Greg Nelson (ed.), *Systems Programming with Modula-3*, Prentice-Hall, 1991.

12] R. Plamonden, C. Y. Suen, and M. Simner (eds) *Computer Recognition and Human Production of Handwriting*, Word Scientific, 1988.

13] Joe M. Pullis, *Speedwriting for Notetaking & Study Skills*, Macmillan, 1990.

14] C. C. Tappert, “Speed, accuracy, flexibility trade-offs in on-line character recognition,”, IBM Research Report RC13228, October 1987.

15] A. J. W. M. P. Thomassen, P. J. G. Keuss, F. P. van Galen (eds), “Motor aspects of handwriting.”, *Acta Psycologica* 54 (1-3), 1983.

16] Harold P. Van Cott and Robert G Kinkade (eds), *Human Engineering Guide to Equipment Design*, U. S. Government Printing Office, 1972.

17] Mark Weiser. “The Computer for the 21st Century,” *Scientific American*, 94-104, Sep 1991.

18] Patricia D. Whitman, “An Evaluation of four shorthand systems after one year of instruction”, Ed.D dissertation, University of Southern California, 1984.