EdgeWrite: A Stylus-Based Text Entry Method Designed for High Accuracy and Stability of Motion

Jacob O. Wobbrock, Brad A. Myers, John A. Kembel
Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213 USA
{ jrock, bam, jkembel }@cs.cmu.edu

ABSTRACT
EdgeWrite is a new unistroke text entry method for handheld devices designed to provide high accuracy and stability of motion for people with motor impairments. It is also effective for able-bodied people. An EdgeWrite user enters text by traversing the edges and diagonals of a square hole imposed over the usual text input area. Gesture recognition is accomplished not through pattern recognition but through the sequence of corners that are hit. This means that the full stroke path is unimportant and recognition is highly deterministic, enabling better accuracy than other gestural alphabets such as Graffiti. A study of able-bodied users showed subjects with no prior experience were 18% more accurate during text entry with EdgeWrite than with Graffiti ($p<.05$), with no significant difference in speed. A study of 4 subjects with motor impairments revealed that some of them were unable to do Graffiti, but all of them could do EdgeWrite. Those who could do both methods had dramatically better accuracy with EdgeWrite.

Keywords
Text entry, text input, unistrokes, Graffiti, Palm, PDAs, handhelds, assistive technology, computer access, motor impairments, corners, edges, gesture recognition, Pebbles.

INTRODUCTION
We are investigating how handheld devices can be used as assistive technologies to provide computer access for people with motor impairments. In our previous work [19], we describe how a handheld device can be used for computer access by people who are unable to use a keyboard and mouse due to motor impairments. People with Cerebral Palsy or Parkinson's often have intention tremor that interferes with use of a mouse [11][20]. People with Muscular Dystrophy often lose their gross motor control and arm strength before losing their fine motor control, making movement between a keyboard and mouse difficult [18][20]. However, moving over the short expanse of a handheld screen often remains feasible. An application we have written called Remote Commander allows people to control their computer entirely from a Palm PDA, providing for both text entry and mouse control. The reader is directed to our previous work for more details [19].

Figure 1. A Palm PDA with an EdgeWrite template over the text input area. A user makes all letters, numbers, and punctuation inside the square hole by moving the stylus along the edges and across the diagonals of the square. Small holes in the template provide access to the four soft buttons on the Palm.

However, text entry on a handheld device is difficult for people with motor impairments. On-screen "soft" keyboards are small and the keys can be difficult to acquire even for able-bodied users. Some research addresses this problem by attempting to discover the "optimal" soft keyboard [30]. Still, for people with motor impairments, these tiny keyboards are difficult to use and consume precious screen space. They also require the user to fix her attention on the keyboard rather than on the output [15]. This is particularly a problem when controlling a PC via a handheld, as with Remote Commander, because the text destination is on the PC monitor, not on the handheld.

Gestural text entry techniques, such as Graffiti, also do not solve the problem of text input for people with motor impairments. Tremor and fatigue dramatically impact a user's ability to make smooth, accurate, and controlled movements [11]. Tremor makes it difficult or impossible

for some subjects to make character forms recognizable by a text entry system like Graffiti (Figure 2) [27].

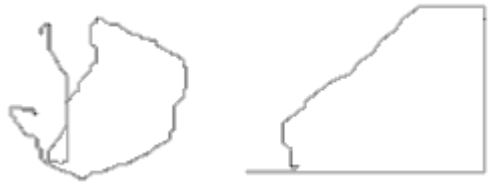

Figure 2. A subject with Cerebral Palsy attempting to make a *d* in Graffiti (left) and EdgeWrite (right). The Graffiti *d* was mis-recognized as an *h*, whereas the EdgeWrite *d* was recognized correctly. The jagged diagonal does not deter recognition in EdgeWrite because only the order in which the corners are hit matters. Note the straight lines where the subject moved along the edges of the plastic template. This EdgeWrite *d* is an alternate form not shown in the character chart in Figure 3.

Another result of tremor is that many users "bounce" the stylus on the screen, triggering unwanted modes and unwanted characters. In fact, this problem renders Graffiti entirely unusable for some people with motor impairments. A more stable means of text entry is necessary for users of handheld devices who have motor impairments.

Able-bodied users will also benefit from more stable means of text entry. Since PDAs are designed to be used "on the go," many situations arise where added stability could be beneficial: riding a bus, walking, or annotating slides during a presentation while standing.

For these reasons, we have been investigating multiple approaches to providing stability through the use of *physical edges* [28]. Edges offer many desirable properties. Appling pressure against an edge while moving a stylus provides:

- *Greater Stability*: Decreased movement variability and movement offset [14].
- *Greater Speed*: Ability to move quickly yet remain on the target line [27].
- *Higher Accuracy*: Targets along an edge or in a corner are easier to acquire [27].
- *Tangible Feedback*: No longer is visual feedback the only means of self-correction during movement, as tactile feedback is available [2].
- *Fitts' Law Benefit*: Edges allow for "target overshoot," where acquiring targets on edges is easier than acquiring targets "in the open" [5][12].

To empirically validate these benefits, we conducted a study of how edges affect the tracing of lines at various angles on a handheld screen by people with motor impairments [27]. We found dramatic advantages for line-tracing along edges versus tracing lines "in the open." Even more dramatic were the advantages of tracing lines that ended in corners. These advantages included the aforementioned stability, speed, and accuracy.

Our research goal is to exploit these "pure" benefits of edges in a text entry technique, and to prevent other factors such as cognitive difficulties from diluting them. We believe our new EdgeWrite[1] input technique goes a long way toward realizing these goals, as it relies heavily on edges and corners, both interactively and algorithmically. These edges are imposed on the input area by a transparent plastic template with a square hole, inside which all text entry is performed (Figure 1). Our empirical results show that extensive iteration of the character set has made the character forms highly guessable and easy to learn, maintaining a low cognitive workload for the user.

In particular, we found that when compared to Graffiti, EdgeWrite was 18% more accurate during text entry for able-bodied users formerly unfamiliar with either technique ($p<.05$). This benefit came without a significant cost in speed. Users with motor impairments succeeded at using EdgeWrite but were largely unable to use Graffiti due to excessive tremor and bounce. The physical edges in EdgeWrite reduced the propensity of tremulous users to bounce the stylus on the screen.

AN OVERVIEW OF EDGEWRITE

EdgeWrite is a stylus-based unistroke input technique. To make a character, a user places the stylus down inside the square hole of the template (Figure 1), moves the stylus in a specific pattern along the edges and diagonals of the square (thereby hitting the corners of the square), and lifts upon completion of the pattern. A character is then entered.

The first difference, then, between EdgeWrite and Graffiti[2] is that all stylus motion in EdgeWrite occurs within a small plastic square hole (1.69 cm^2), which bounds the input area with hard physical edges.

A second difference between EdgeWrite and Graffiti is that recognition does not depend on the *path* of movement, nor is the recognizer a pattern matcher. Instead, recognition only depends on the *order in which the corners are hit*. The advantages of this include:

- Users with motor impairments can "wiggle" or slide and this does not degrade recognition.
- The recognition algorithm is elegant and fast, as hit-testing corner areas is an operation capable of being performed rapidly by a weak processor.
- Users can teach the system their own custom gestures with *one* example, as training sets for a pattern matching algorithm are not necessary.
- For the designer, it is easy to iterate character forms; changing one only requires changing a corner sequence value (a single integer). No sets of points or glyphs are necessary.

[1] U.S. patent pending.

[2] This paper refers often to Graffiti [7] because of its familiarity. Most of our comments can be fairly applied to Unistrokes [6] and Jot [9] as well.

A third difference between EdgeWrite and Graffiti is the reduction of modes in EdgeWrite. In particular, EdgeWrite uses no shift, caps lock, or extended shift modes. The only mode in EdgeWrite is a punctuation mode.

An important question is whether the EdgeWrite character forms are easy to guess, learn, and make. Prior research shows that Graffiti characters have these properties for able-bodied users [16]. The evaluation discussed at the end of this paper shows these properties to be true of EdgeWrite as well.

Figure 3 is the character chart for current version of the primary character forms in EdgeWrite. In addition, multiple alternate forms exist for nearly every character (not shown). The character forms are a product of many hours of user testing and extensive iteration. In user testing, many people discovered and used several of the alternate character forms despite their absence from the chart, suggesting a high degree of guessability.

Though many of the EdgeWrite characters look vaguely like their handwritten counterparts, the mnemonic power of these characters comes less from their appearance and more from their "feel." One subject noted this when, after entering 20 phrases in EdgeWrite, he said, "I don't remember any of the pictures in my mind, but I still *feel* them in my hand."

As in Graffiti, some gestures resemble lowercase letters, while other gestures resemble uppercase letters. All gestures produce lowercase letters unless the capitalization *suffix stroke* is appended to the gesture. The suffix stroke is simply a motion to the upper-left corner (think "up" to "make it big") *after* the regular gesture is made but *before* lifting the stylus. Note that, by design, no gestures finish in the upper-left corner, allowing for this modeless design. In user studies, subjects had no trouble with this method of capitalization.

Primary EdgeWrite Character Forms

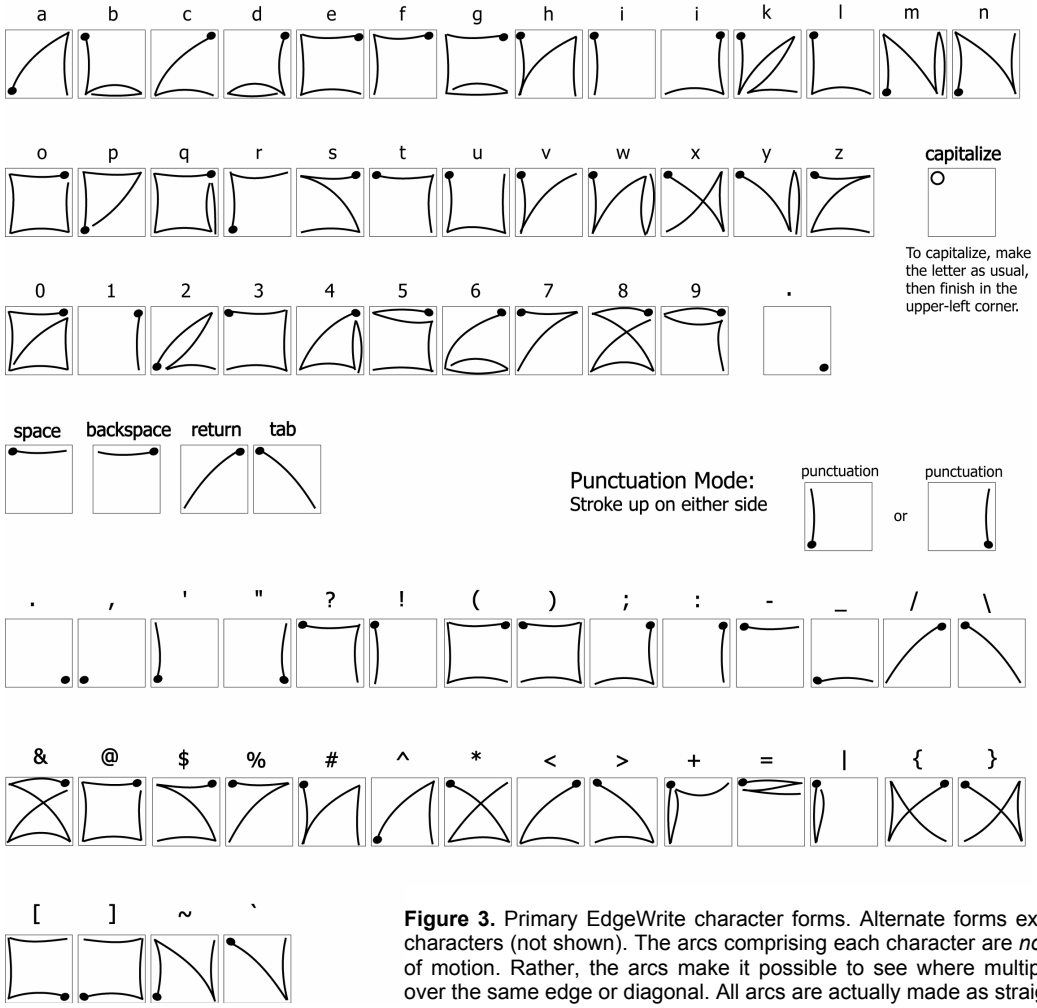

Figure 3. Primary EdgeWrite character forms. Alternate forms exist for nearly all characters (not shown). The arcs comprising each character are *not* the literal path of motion. Rather, the arcs make it possible to see where multiple strokes pass over the same edge or diagonal. All arcs are actually made as straight lines.

Another thing to notice about the character chart is that it is *representational, not literal*. We faced a design challenge in representing the strokes on paper, as many characters have strokes that pass over the same edge or diagonal more than once. If such a "double pass" is drawn literally, then the result is merely a single line. Following a suggestion from a professor of visual design, we chose to arc the paths into the intended corners (Figure 4). These arcs make it possible to clearly depict a double-pass. Our subjects had little trouble reading the chart once we explained this to them. In EdgeWrite, all movements are, in the ideal case, straight lines. However, as mentioned, straight line motion is not necessary for recognition, only hitting the corners in the proper order.

Further detail about the design and implementation of EdgeWrite is saved for a later section. Now we turn to the related work that informed the creation of EdgeWrite.

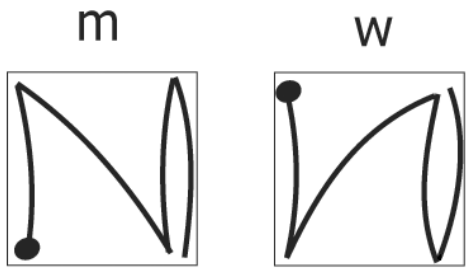

Figure 4. These letters show how arcs reveal strokes that pass over the same edge more than once, yet still indicate the intended corner sequence. Both of these letters use the technique of collapsing the letters' right sides onto one edge.

RELATED WORK

The related work falls into four areas: (1) text entry techniques, (2) assistive technologies and modeling of motor-impaired performance, (3) text entry analysis techniques, and (4) using edges to exploit Fitts' Law.

Text Entry Techniques

Related techniques for entering text on handhelds come in two main forms: on-screen soft keyboards and gesture-based methods.

As mentioned in the introduction, we employed soft keyboards in Remote Commander but found them difficult for some users with motor impairments. Able-bodied users generally find them fast, however, so there has been an effort to create optimized keyboards which minimize travel distance between letters through quantitative modeling [30]. Many other soft keyboard designs exist: a nice overview is provided in [15]. One problem with all of them, however, is the increased focus-of-attention incurred because users have to look at the keyboards while using them.

More similar to EdgeWrite are the gestural text entry techniques, which have a history dating back to as early as 1957 [3]. Unistroke methods, for example, separate characters during text entry by pen-down/pen-up sequences. The term "unistroke" originated from the alphabet by the same name developed at Xerox PARC [6]. However, Unistrokes did not resemble real letters, and for this reason were difficult to learn and recall [15]. Graffiti used character forms similar to handwritten forms and proved much easier to retain than Unistrokes [16]. A later unistroke research effort discovered that the easiest strokes to make on a variety of devices were in the four cardinal directions, so a "device independent" alphabet was created using them [8]. EdgeWrite naturally employs a high percentage of strokes in the four cardinal directions.

The character recognition approach in EdgeWrite is similar to that used in LibStroke [25] and Xstroke [29], as all three use hit-testing of geometric regions, though EdgeWrite is the only one to rely on edges and corners. Plastic templates have been used in input techniques on touch-sensitive graphical tablets for some time [2]. EdgeWrite is the first system of which we are aware to combine these two concepts in a text entry technique.

In contrast to unistrokes, *continuous* gesture techniques do not require lifting the stylus between characters in order to improve the speed of input. Rather than making character forms, the user moves the stylus through different regions, and segmentation between letters is accomplished by exiting one region and entering another. Examples are Quikwriting [22], Cirrin [17], and Edge Keyboards [28]. The latter two have the same problem as soft keyboards (increased focus-of-attention) because they require constant visual attention.

Though EdgeWrite in its current form is a unistroke technique, we are actively exploring ways to create a continuous gesture version. EdgeWrite already has the elevated physical edges imposed by the plastic template to provide tactile feedback and positional information without visual cues [2]. If the user does not have to lift the stylus, it may be possible to enter text completely eyes-free. This has positive implications for the use of handhelds by people who are blind, or by able-bodied people in eyes-free situations, such as taking notes while observing an object of interest.

Assistive Technologies for Motor Impairments

Researchers at Cambridge University have modeled motor-impaired cursor motion to improve user models and input techniques [10][11]. Relevant assistive technologies aimed at providing computer access include the Stanford Archimedes Project [21], which uses a custom handheld device, as well as numerous assistive technologies for text entry: row-column scanning, mouse-driven keyboards, word prediction systems, and one-handed keyboards [1].

Text Entry Analysis Techniques

Until recently, most text entry studies artificially constrained input because of a lack of automated analysis techniques for errors in uncorrected text. While speed is straightforward to measure, accuracy—both *during* text

entry and in the resultant string—are more difficult [24]. Space precludes a discussion of text entry analysis here, but our evaluation of EdgeWrite is indebted to the work of those who have developed improved analysis algorithms for evaluating text entry. In particular, the Levenshtein minimum string distance statistic [24] is helpful, as well as the dependent measure of keystrokes per character (KSPC) [13][24], which we extend to gestures per character (GPC), described below.

Using Edges to Exploit Fitts' Law

Fitts' Law predicts the time it takes to acquire a target from the size of the target and the travel distance to the target. Specifically, the acquisition time is logarithmically related to the travel distance and the target size [5].

While Fitts' Law has been studied extensively [12], it has been exploited by using *edges* relatively little in user interfaces. Exceptions include a Web browser that placed the "back" button at the left edge of the screen and found reduced acquisition time [4]. Another exception is the Macintosh user interface [26], which places menus atop the screen. Other work has looked at keyboard layouts placed around the edges of PDA screens, where the device chassis provides an elevated edge against which a stylus can hit [28].

THE DESIGN AND ENGINEERING OF EDGEWRITE

This section gives more detail about the design and implementation of EdgeWrite. Because EdgeWrite is the product of a highly iterative process, the evolution of key facets is worth some discussion.

The Feel of EdgeWrite Characters

We believe it is because of the Fitts' Law benefits of edges—the ability to overshoot a target by an arbitrary amount—that certain EdgeWrite characters "feel like" their handwritten counterparts (Figure 5). When a user hits a corner with a high force as she moves her stylus, she may feel as though she is moving farther in that direction than when she hits the same corner with less force. We intend to study this phenomena in more detail as part of our future work, but our observations of many subjects suggest that this is precisely why EdgeWrite characters can be easily learned: they feel similar to real characters despite being mapped onto a square.

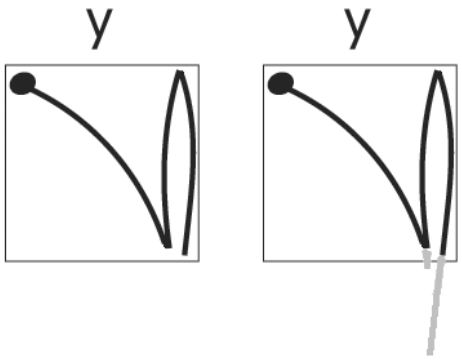

Figure 5. The form for *y* impacts the lower-right corner twice: once on the diagonal down-stroke, and once on the straight down-stroke (left). However, in making this form, it can "feel like" the second stroke goes farther down because of a greater force applied to the bottom edge on the second impact (right).

In Defense of a Square

The choice of a square hole was not arbitrary in the design of EdgeWrite. For one, only the use of a square guarantees that the edge-following motions will be in the four cardinal directions: up, down, left, and right. These have been shown to be the easiest directions to move in a gestural alphabet, even without an edge [8][15].

Another reason we chose a square is for simplicity. A more complicated polygon would result in more vertices and more possibilities for the next corner to be hit while moving. This could be cognitively overwhelming in the case of many proximate vertices. Higher-order polygons also would result in more oblique angles at the vertices, and these may cause the stylus to "slip out" during movement. While an acute angle pockets a stylus best, a square's 90° corners are adequate to catch a fast-moving stylus. By contrast, a triangle would too severely limit the number of possible character forms and reduce the extent to which the forms resemble their handwritten counterparts.

If we define a "segment" to be a straight line stroke between two vertices (corners), then for gestures made inside a closed shape with v vertices, the number of possible character forms using s segments is given by the formula:

$$forms = \sum_{i=0}^{s} v \cdot (v-1)^i$$

This formula treats a tap at a vertex as a legal stroke, and assumes that the same corner is never used twice in a row.

For a square, $v = 4$. If $s = 0$, meaning we use no segments, we see that we have 4 possible forms available to us: a tap in each of the four corners. With 1 segment we get 16 possible forms ($4 + 4 \times 3$), with 2 we get 52 forms, and with 3 we get 160 forms. Thus, there is a wealth of forms to choose from with relatively few segments.

The character chart in Figure 3 represents 100 characters: 26 lowercase letters, 26 uppercase letters via the capitalization suffix stroke, 10 digits, 4 white space characters, 2 punctuation mode-setters, and 32 punctuations. We do not count period twice, as it is the same form in and out of punctuation mode. Not pictured in the chart are the four directional arrow keys, which are also implemented, making for 104 *unique* characters in the current EdgeWrite set. Incidentally, there are 288 forms in the *entire* set, which includes alternate forms.

The average primary EdgeWrite character form has 2.47 segments in it, excluding capitalization. If we include capitalization and its associated suffix stroke, this average increases to 2.84. The average number of segments per character for the entire EdgeWrite character set, including all alternates and capitals, is 3.49.

Since we have 102 characters excluding punctuation mode-set, the *forms* equation above dictates that we *must* use 3 segments for at least some of the characters—50 of them to be exact. If we designed the character set with the fewest number of segments possible and no modes, and with only one form for each character, then the average number of segments per character in that set would be 2.39. So even with high learnability and guessability, the average segments per primary character in EdgeWrite (2.84) is less than ½ segment longer than the theoretical lower bound (2.39). For entry without capitals (e.g., instant messaging), the average is even closer (2.47).

The current size of the square hole in the EdgeWrite template is 1.69 cm^2. This size was chosen to strike a balance between competing constraints. We wanted to shorten the distance of movement as much as possible, particularly for those with motor impairments. On the other hand, the corner regions needed to be spacious enough to be reliably hit by the user, but not too close together or they would be hit accidentally. Based on the thickness and the size of the of the Palm stylus tip, we settled on the current square size. After a good deal of user testing, we have seen little reason to believe that this size should change for the PDA version of EdgeWrite.

Evolution of the Corners

Though EdgeWrite's corner-based recognition algorithm is straightforward, the design of the corners themselves was not obvious. The corners began naively as points rather than areas, and this proved to be inadequate, as users rarely hit the exact pixel in the corners. This was because users held their styluses at various angles. An angled stylus impacts the edge of the plastic template a few millimeters above its tip, causing the tip to jut into the square a few pixels even when the stylus is flush against the edge (Figure 6).

After we increased the corner size to an appreciable area, two other problems emerged. Once moving, users would often accidentally hit corners, particularly when doing a diagonal stroke, as in an *s*. However, if the corners were too small, users would often fail to hit them on pen-down, particularly in the backspace stroke (across the top or bottom from right to left). It seemed that we needed large corners for when the stylus went down, but then small corners thereafter.

The next step in the design process added precisely this: We *inflated* the corners until the stylus was detected within one of them, then *deflated* all of them while the stylus was moving (Figure 7). Deflated corners are not rectangular but are triangular to make accidental corner-hitting even rarer, especially during the making of a diagonal segment.

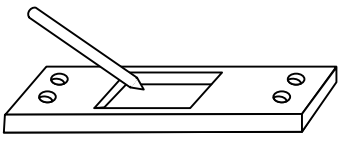

Figure 6. The tip of an angled stylus meets the screen a small distance from the edge of the dominant-hand-side of the square.

An observation during a user study prompted the most recent iteration on the corners. A right-handed user with a chronic wrist injury held the stylus at a fairly shallow angle relative to the PDA screen (e.g., Figure 6). The result was that the elevated edge of the plastic square prevented the tip of the stylus from getting close to the right side of the square. Figure 7 shows our adjustments for both right- and left-handed users. We provide extra corner area along the *x*-axis for the dominant-hand side of the square in order to account for users who hold their styluses at steep angles. A nice property of this design is that it does not negatively impact users who hold their styluses more vertically.

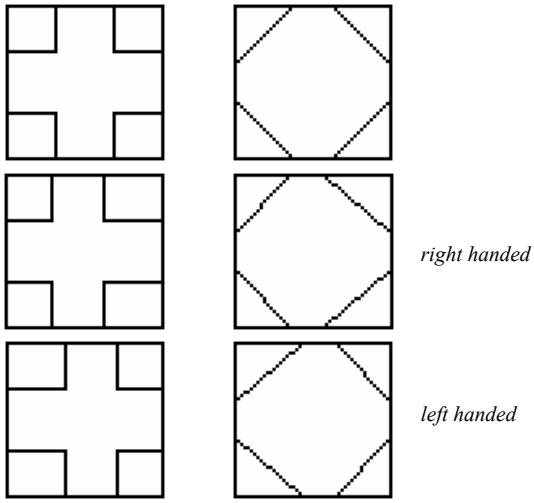

Figure 7. The left column shows what the EdgeWrite recognition algorithm regards as the corners *before* the stylus is detected inside one of them. Once the stylus enters a corner area, the corners deflate for the duration of the stroke, shown in the right column. Note that "deflation" is not only in size but also in shape.

Lessons Learned from Graffiti

Though Graffiti is as close to a "killer app" as any handheld application, it is not perfect. We examined prior studies of Graffiti (e.g., [16][23]) in an effort to improve upon some of Graffiti's problems.

Certain letters in Graffiti have specific problems. For example, many people handwrite an *n* beginning at the top-left of the letter and going down. In Graffiti, this nearly always results in a *w* or *h* [16]. EdgeWrite supports an optional initial down stroke on letters that commonly have them: *d*, *m*, *n*, *p*, *q*, and *r*. Similarly, many people make a down stroke at the end of *u*, and in Graffiti this nearly always produces an *h* [16]. EdgeWrite allows this down stroke on *u*. Graffiti also often produces a *u* when novices make a *v* but forget to add the serif on the right. EdgeWrite avoids this problem, as every form is more than just *subtly* different from every other form.

Another problem for novices using Graffiti is confusion between the *x* and *k*, as these are mirror images of each other [16]. EdgeWrite removes this similarity by redesigning the *k* so that it starts at the top-left, where a handwritten *k* starts, not at the top-right, where a Graffiti *k* starts. (We leave the top-right *k* as an alternate form for habituated Graffiti users.)

As mentioned above, users with motor impairments sometimes "bounce" inadvertently on the screen. An older version of EdgeWrite had characters that were entered by taps in the corners. We removed all of these except period (.) to reduce the likelihood of entering an accidental mode or character by inadvertent taps on the screen.

In EdgeWrite, we differentiate position based on the known location of the square. Hence, we can distinguish *i* from *1* even though they are both downward strokes because they are on different edges. This is a powerful concept, as it allows for input in a very small area: EdgeWrite does not need separate regions of the screen for letters, numbers, capitals, and so on.

Implementation

The implementation of EdgeWrite enables fast character recognition. The recognizer does nothing until it detects a pen-down event in one of the corners. Once it does, it begins queuing up all the points over which the stylus moves until the stylus is lifted. No recognition or filtering is done during the stylus movement in order to maximize the number of movement points queued. Once the stylus is lifted, the recognizer processes the point queue and hit-tests the points against the corner regions. The result of this loop is a 32-bit integer value representing the sequence in which the corners were hit. This integer is assembled efficiently: when a new corner is hit, the existing integer sequence is shifted to the left and the new corner is appended. This sequence is then sent to a lookup function that finds the character corresponding to the corner sequence, if any.

This recognition algorithm is fast in linear time $O(n)$, and we believe it could be implemented on a weak processor with a poor digitizer sampling rate and a noisy digitization of stylus coordinates. Anecdotally, it was not possible for us to move the stylus faster than EdgeWrite could recognize the stroke on a Palm V, which polls its screen for the pen every 20 milliseconds.

The "other part" of EdgeWrite is implemented not in software but in plastic. The template is crucial for EdgeWrite to work well, and designing and fabricating this plastic piece was just as iterative as developing the software. We have numerous prototypes, and two recent models are shown in Figure 8. Model 8*a* is small and sits on the Palm screen. We found this to work fine for able-bodied users, but users with motor impairments sometimes put pressure with their fingers on the template, causing it to press against the Palm screen and confuse the digitizer. We designed model 8*b* in order to avoid putting pressure on the screen: it sits on the Palm chassis and therefore cannot touch the screen (Figure 1). Anchoring the template in an elegant and robust way is a design challenge for future work. For now, we just tape it to the edges of the Palm case, which works fine.

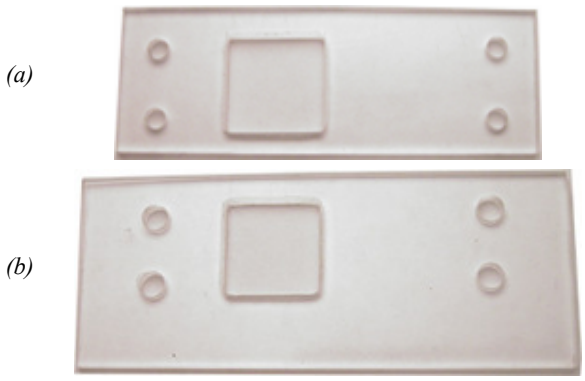

Figure 8. Two transparent plastic templates (*actual size*). Both are anchored over the normal text entry area of a Palm PDA using tape. Template (*a*) sits in contact with the Palm screen and, if pressed upon, can interfere with the digitizer. Template (*b*) avoids this problem by sitting on the Palm chassis, above the screen a few millimeters (see Figure 1).

AN EVALUATION OF EDGEWRITE

We conducted two evaluations of EdgeWrite: a formal study with able-bodied users, and some observational studies with 4 motor-impaired users.

Study #1: EdgeWrite vs. Graffiti, Able-Bodied Users

Though EdgeWrite is targeted towards people with motor impairments, we wanted to assess its speed and accuracy with able-bodied users to validate its design.

Procedure: Subjects, Tasks, Apparatus, Measures

We recruited 10 paid subjects who had no prior experience with handheld text entry. Each subject was assigned randomly to a between-subjects condition for *input method*: Graffiti or EdgeWrite. The experiment lasted 1 hour and consisted of two parts: a practice session followed by a testing session.

Before the practice session, we taught subjects the basics of the input method to which they were assigned. For Graffiti, this included things like pointing out the area of input, pointing out the letter area and the number area, and demonstrating the appropriate pressure with which to hold the stylus against the screen. It did not include expert-level nuances like making the *v* backwards. For EdgeWrite, this explanation included the importance of pressing firmly against the edges during movement, and the importance of hitting corners in the correct order.

The practice session consisted of making each character *twice successfully*. Thus, if a character was not recognized correctly, it was retried until success. Subjects progressed through a character chart for EdgeWrite or Graffiti, a subset of the characters shown in Figure 3.

The testing session consisted of entering 20 sentences, all about 50 characters in length. Task sentences were

presented in the same fixed order for every subject. The sentences appeared on the Palm PDA screen, just above the text entry area where subjects reproduced the text. The close proximity of the task sentence and the text entry area meant reduced focus-of-attention and prevented unwarranted errors.

The character chart was available to subjects for use if they could not remember how to make a character. Subjects were encouraged to guess "once or twice" if they could not recall a character, after which they could use the chart. Their uses of the chart were recorded.

Subjects were not artificially constrained in their entry of text: they could make errors or get out of sync with the task sentence. They were told before the first task to "proceed quickly and accurately, as you would enter text in the real world" [24].

We designed the tasks to resemble realistic entries: letters, names, phone numbers, addresses, and URLs [23]. We felt these tasks would yield a more realistic measure of speed and accuracy, as opposed to measures resulting from tests of only lowercase letters, for example.

All task and performance data was logged on a nearby laptop over a serial cable attached to the Palm PDA. This data was then processed by a log file parser capable of computing numerous statistics like task times from pen-down and pen-up sequences, intra-character times, errors in the final string, keystrokes per character, gestures per character, etc. Space precludes a detailed discussion of each measure. The interesting ones are discussed below.

Results

The three measures of obvious importance are speed, accuracy *during* text entry, and accuracy *after* text entry. We calculate all three of these measures with the methods from [24].

Subjects continually improved over the first 12 tasks, then began to flatten out in speed and accuracy, so here we treat these first 12 tasks as training exercises and consider only the last 8 tasks in our comparisons. *In all comparisons, the EdgeWrite value is reported first, followed by the Graffiti value.*

A one-way ANOVA for input technique with repeated measures for task yields a non-significant result for entry speeds: 6.6 to 7.2 wpm ($F[1,56]=.27$, n.s.). If we consider all 20 trials to examine learning, the means are even closer: 5.85 to 5.97 wpm ($F[1,152]=.02$, n.s.). Further testing is required to differentiate the speeds of these two methods.

Keystrokes per character (KSPC) is the ratio of the number of characters entered while transcribing a string to the number of characters in the resultant string [13][24]. If every character is entered correctly the first time, no corrections are made and a perfect score of 1.0 is the result (lower is better). The ANOVA yields a significant result in favor of EdgeWrite: 1.21 to 1.43 KSPC ($F[1,56]=9.32$, $p<.02$), an 18% improvement. Note that we did not include non-character-producing strokes such as mode-setters in this measure, as that would unduly penalize any entry technique with modes. Figure 9 shows KSPC for Graffiti and EdgeWrite over the last 8 tasks.

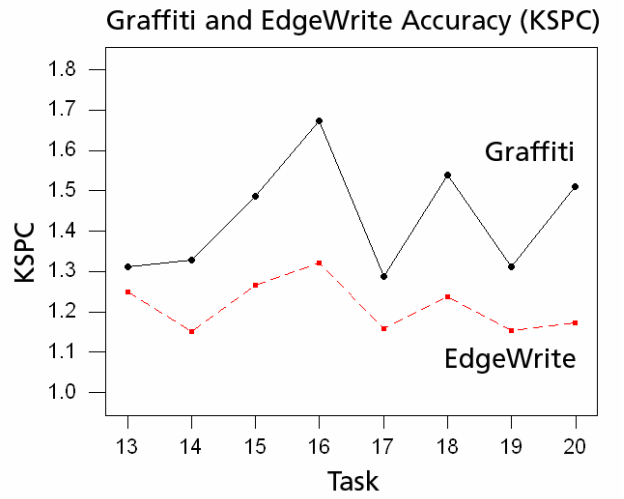

Figure 9. Keystrokes per character as a measure of accuracy during input. Lower KSPC is better. A perfect score is 1.0.

Errors left in the transcribed string can be compared to the task sentence using the Levenshtein minimum string distance statistic [24]. This computes the percentage of errors between two strings based on how many editing primitives (insertion, substitution, deletion) would be required to transform one string into the other. The ANOVA yields a non-significant result in favor of EdgeWrite: .34 to .39 *percent* errors ($F[1,56]=.02$, n.s.). For our tasks, this is roughly equivalent to one erroneous character for every 250 characters entered (about 5 tasks).

We also can examine the intra-character time (pen-down to pen-up) to determine which alphabet's strokes are faster. The ANOVA yields a significant result in favor of Graffiti: .58 to .29 seconds ($F[1,56]=20.57$, $p<.005$). Note that this did not translate into faster entry speed (wpm) for Graffiti, as error correction and thinking time play a part.

We introduce a new measure similar to KSPC called gestures per character (GPC). This measure is the same as KSPC but includes mode setters and all other *recognized* strokes—things KSPC must leave out. The result can be used to show which alphabet is more "verbose," requiring more multi-stroke characters and more modes. The ANOVA for GPC yields a significant result in favor of EdgeWrite: 1.26 to 1.62 GPC ($F[1,56]=14.65$, $p<.005$). This verifies our claim that EdgeWrite uses less mode strokes, something important for motor-impaired users, with whom every stroke counts.

Finally, *both* alphabets were extremely learnable and guessable. For EdgeWrite, subjects had to use the character chart for an average of 4.6 *total* characters out of 1000 characters entered. For Graffiti, this average went

up to 6.6. This comparison did not yield a significant difference (F[1,152]=1.72, n.s.).

Another measure of learnability is the amount of time spent on each task. Figure 10 shows that EdgeWrite and Graffiti compared similarly. Overall means for task time were nearly identical, at 118.16 seconds for EdgeWrite to 117.12 seconds for Graffiti (F[1,152]=0.0, n.s.).

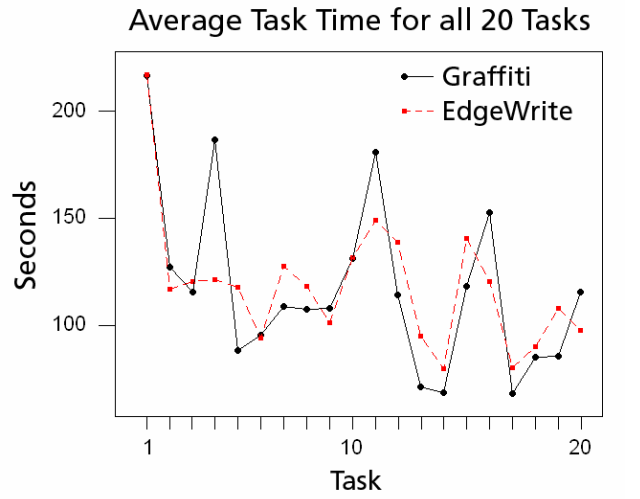

Figure 10. Average task times across all 20 tasks show that EdgeWrite and Graffiti were learned at comparable rates. The upward spikes in the downward trend are due to tasks with higher amounts of punctuation interspersed among simpler tasks.

Study #2: EdgeWrite vs. Graffiti, Motor-Impaired Users

Obtaining statistical results with subjects who have motor impairments is often infeasible due to high variance among people with disabilities, and because of the difficulty in obtaining subjects. We observed 4 users with motor impairments interacting with both EdgeWrite and Graffiti and logged their data. Error counts are given as the minimum string distance [24].

Subject 1

Subject 1 was a middle-aged woman with Parkinson's Disease and severe tremor. We asked her to enter each lowercase letter (a-z) and each digit (0-9) twice in Graffiti and then twice in EdgeWrite. Her recognition rates were 22/72 and 68/72, respectively. She used the character chart during the whole exercise. In Graffiti, interspersed with the mis-recognitions were a number of accidental periods and other punctuation due to "bounce" on the screen. Her tremor made it difficult for her to *avoid* setting the punctuation mode in Graffiti. When she entered a 50 character sentence in Graffiti and EdgeWrite, she made over 3 times the number of errors in Graffiti.

Subject 2

Subject 2 was a 30-something woman with Spastic Cerebral Palsy. When asked to make each letter and number on the Graffiti chart, it turned out there were some she could never correctly produce despite repeated efforts: *c*, *e*, *f*, *x*, *2*, *4*. For EdgeWrite, these were limited to *2* and *7*. When she attempted the sentence, "The dog is going fast," in Graffiti she produced, "The g i gbsiangu% fast" (8 errors). She skipped *d* after repeated mis-recognitions because it was too difficult. In EdgeWrite, she entered the sentence without leaving any errors. Her only complaint was that the diagonals were difficult.

Subject 3

Subject 3 was a 30-something male with a form of Muscular Dystrophy. He had a small baseline tremor. Using Graffiti, he was unable to make *b*, *d*, and *f* after multiple tries and gave up on them. Using EdgeWrite, there were not any letters that he could not produce. In fact, in an attempt to do 3 versions of each EdgeWrite character, he made only 2 errors in 108 characters. When asked to produce the sentence, "the ugly underwater urchin eats putrid meat lovingly," in Graffiti he produced, "the ugl un erwater urchin eats putri meat lovingly" (3 errors). In EdgeWrite he produced the sentence without leaving any errors.

Subject 4

Subject 4 was a 40-something woman with Cerebral Palsy. She was able to enter all letters and numbers in both Graffiti and EdgeWrite, but upon switching to EdgeWrite from Graffiti, she exclaimed, "This is a lot easier than before. Much easier, my goodness." She entered a 50 character sentence in Graffiti leaving 2 errors and in EdgeWrite without leaving any errors.

FUTURE WORK

We are pursuing the development of a continuous entry method, which would allow for eyes-free entry and could be used for visually-impaired people. We are also building versions of EdgeWrite for other platforms: cell phones, watch faces, Trackpads, PocketPCs, and joysticks. We would like to study text entry with joysticks on game consoles, comparing EdgeWrite to current game console methods. We plan to run other studies as well: expert transition from Graffiti to EdgeWrite, extended use over long periods of time, retention of the alphabet after a period of non-use, guessability of character forms, and text entry in unstable environments, such as while walking down the street or standing on a bus. We also require more validation with motor-impaired users. Finally, we would like to begin developing models of motor-impaired stylus behavior in hopes of gaining insights for the design and refinement of input techniques.

CONCLUSIONS

We have shown that physical edges can be an effective means of providing stability in stylus input techniques. We have also shown that a unistroke alphabet comprised entirely of strokes along 4 edges and 2 diagonals can be highly guessable and learnable, despite the natural concern that such a constrained alphabet would not be usable. We have provided an algorithmic approach to character recognition based on hit-testing in dynamic size-and-shape-changing corner regions. This approach to recognition resulted in an 18% accuracy improvement

during text entry for EdgeWrite over Graffiti, without significant detriment to entry speed.

EdgeWrite's recognition technology could be implemented on other platforms, including those without the luxury of a fully-digitized touch screen. All that is required is 4 corner sensors and 1 surface sensor. These sensors could be crude: they do not have to determine coordinates, only whether a stylus or finger is in contact with them or not. Thus, we have shown that a reliable character recognizer need not be a pattern matcher that depends on the whole path of movement.

We are hopeful that many domains will find EdgeWrite or the EdgeWrite recognition approach useful, from PDAs to game consoles to assistive technologies.

ACKNOWLEDGMENTS

The authors thank Connie Campbell and the UCP of Pittsburgh, Aaron Bauer, Victoria Danforth, Darren Gergle, Silvia Kembel, Andrew Ko, Jeffrey Nichols, Alison Wobbrock, and John Zimmerman. This work was supported by the NEC Foundation, General Motors, Microsoft, and the National Science Foundation. For more information and software downloads, see http://www.cs.cmu.edu/~pebbles/assistive/

REFERENCES

1. Anson, D.K. *Alternative Computer Access.* F. A. Davis Co., 1997. Chapters 8 and 14.
2. Buxton, W., Hill, R., Rowley, P. Issues and techniques in touch-sensitive tablet input. *Computer Graphics 19* (3), 1985, 215-223.
3. Dimond, T.L. Devices for reading handwritten characters. *Proc. Eastern Computer Conference*, 1957, 232-237.
4. Farris, J.S., Jones, K.S., Anders, B.A. Acquisition speed with targets on the edge of the screen: An application of Fitts' Law to commonly used Web browser controls. *Proc. Human Factors and Ergonomics Society 45th Annual Meeting*, 2001, 1205-1209.
5. Fitts, P.M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47* (6), 1954, 381-389.
6. Goldberg, D., Richardson, C. Touch typing with a stylus. *Proc. INTERCHI '93*, 1993, 80-87.
7. Graffiti. Palm, Inc. Available at http://www.palm.com/products/input/
8. Isokoski, P. A minimal device-independent text input method. Unpublished thesis, University of Tempere, Finland, 1999.
9. Jot. Communication Intelligence Corporation (CIC). Available at http://www.cic.com/products/jot/
10. Keates, S., Clarkson, P.J., Robinson, P. Investigating the applicability of user models for motion-impaired users. *Proc. ASSETS '00*. ACM Press, 2000, 129-136.
11. Keates, S., Hwang, F., Langdon, P., Clarkson, P.J., Robinson, P. Cursor measures for motion-impaired computer users. *Proc. ASSETS '02*. ACM Press, 2002, 135-142.
12. MacKenzie, I.S. Fitts' Law as a research and design tool in human-computer interaction. *Human-Computer Interaction 7* (1). Lawrence Erlbaum, 1992, 91-139.
13. MacKenzie, I.S. KSPC (keystrokes per character) as a characteristic of text entry techniques. *Proc. Mobile HCI '02*. Springer-Verlag, 2002, 195-210.
14. MacKenzie, I.S., Kauppinen, T., Silfverberg, M. Accuracy measures for evaluating computer pointing devices. *Proc. CHI '01*. ACM Press, 2001, 9-15.
15. MacKenzie, I.S., Soukoreff, R.W. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction 17* (2). Lawrence Erlbaum, 2002, 147-198.
16. MacKenzie, I.S., Zhang, S. The immediate usability of Graffiti. *Proc. Graphics Interface '97*. Canadian Information Processing Society, 1997, 129-137.
17. Mankoff, J., Abowd, G.D. Cirrin: A word-level unistroke keyboard for pen input. *Proc. UIST '98*. ACM Press, 1998, 213-214.
18. MDA. The Muscular Dystrophy Association, 2001. Available at http://www.mdausa.org/
19. Myers, B.A., Wobbrock, J.O., Yang, S., Yeung, B., Nichols, J., Miller, R. Using handhelds to help people with motor impairments. *Proc. ASSETS '02*. ACM Press, 2002, 89-96.
20. Neurology Channel. Movement disorders. Available at http://www.neurologychannel.com/movementdisorders/
21. Newman, Keith. The open interface: Beyond keyboards and mice. *e.nz Magazine*, May/June 2002, 6-11. Available at http://archimedes.stanford.edu/Archimedes.pdf
22. Perlin, K. Quikwriting: Continuous stylus-based text entry. *Proc. UIST '98*. ACM Press, 1998, 215-216.
23. Sears, A., Arora, R. Data entry for mobile devices: An empirical comparison of novice performance with Jot and Graffiti. *Interacting with Computers 14* (5). Elsevier Press, October 2002, 413-433.
24. Soukoreff, R.W., MacKenzie, I.S. Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic. *Extended Abstracts CHI '01*. ACM Press, 2001, 319-320.
25. Willey, M. Design and implementation of a stroke interface library. *IEEE Region 4 Student Paper Contest*, 1997. Available at http://www.etla.net/libstroke/libstroke.pdf
26. Williams, G. The Apple Macintosh computer. *Byte 9* (2), 1984, 30-54.
27. Wobbrock, J.O. The benefits of physical edges in gesture-making: Empirical support for an edge-based unistroke alphabet. *Extended Abstracts CHI '03*. ACM Press, 2003, 942-943.
28. Wobbrock, J.O., Myers, B.A., Hudson, S.E. Exploring edge-based input techniques for handheld text entry. *Third Int'l Workshop on Smart Appliances and Wearable Computing (IWSAWC '03)*. In *Proc. 23rd IEEE Conference on Distributed Computing Systems Workshops (ICDCS '03)*. IEEE Press, 2003, 280-282.
29. Worth, C.D. Xstroke: Full-screen gesture recognition for X. *Proc. USENIX '03*, 2003, 187-196.
30. Zhai, S., Hunter, M., Smith, B.A. Performance optimization of virtual keyboards. *Human-Computer Interaction 17* (3). Lawrence Erlbaum, 2002, 229-269.