Trackball Text Entry for People with Motor Impairments

Jacob O. Wobbrock and Brad A. Myers
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{ jrock, bam }@cs.cmu.edu
http://www.edgewrite.com/dev.html

ABSTRACT
We present a new gestural text entry method for trackballs. The method uses the mouse cursor and relies on *crossing* instead of pointing. A user writes in fluid Roman-like unistrokes by "pulsing" the trackball in desired letter patterns. We examine this method both theoretically using the Steering Law and empirically in two studies. Our studies show that able-bodied users who were unfamiliar with trackballs could write at about 10 wpm with <4% total errors after 45 minutes. In eight sessions, a motor-impaired trackball user peaked at 7.11 wpm with 0% uncorrected errors, compared to 5.95 wpm with 0% uncorrected errors with an on-screen keyboard. Over sessions, his speeds were significantly faster with our gestural method than with an on-screen keyboard. A former 15-year veteran of on-screen keyboards, he now uses our gestural method instead.

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User interfaces — *Input devices and strategies*.
K.4.2. Computers and society: Social issues – *assistive technologies for persons with disabilities*.

Author Keywords
Text entry, text input, gestures, unistrokes, trackballs, pointing, crossing, Fitts' Law, Steering Law, EdgeWrite.

INTRODUCTION

Trackballs are the preferred pointing devices for numerous computer users, particularly for people with some form of motor impairment [9]. For people with low strength, poor coordination, wrist pain, or limited ranges of motion, rolling a trackball (Figure 1a) can be easier than shuttling a mouse across the surface of a desk.

Not surprisingly, many people who prefer trackballs due to motor impairments cannot use a conventional physical keyboard. For these people, some form of text entry besides typing is required. A common solution is to use a trackball with an on-screen keyboard and to click or dwell on the virtual keys. But this method of text entry has many problems. For instance, on-screen keyboards require precise pointing over small targets, a difficult task in itself and particularly so for trackballs, which can have a hard time remaining over targets [14]. Also, on-screen keyboards consume precious screen real-estate. They exacerbate mouse travel to and from a document, particularly for editing [28]. Furthermore, they require two foci-of-attention, one on the document and one on the keyboard. Although physical keyboards can be used by feel, an on-screen keyboard can only be used in a hunt-and-peck fashion by sight. This intense reliance on sight makes on-screen keyboards tedious and visually taxing.

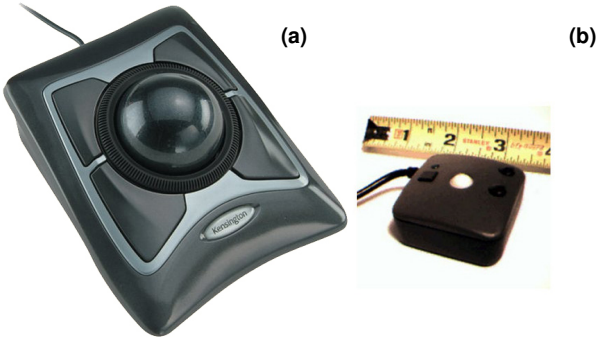

Figure 1. (a) A Kensington Expert Mouse trackball, one of the largest trackballs on the market and popular among people with motor impairments. **(b)** An Appoint Thumbelina, one of the smallest trackballs ever made.

Although speech recognition can be a viable solution for some, it has known drawbacks and limitations and can be unsuitable for people with certain impairments [13]. What is needed is a trackball text entry method that leverages the strengths of trackballs and does not require an on-screen keyboard. Putting the text entry method on the trackball itself reduces physical movement among devices and the need for multiple devices to be within reach [21]. It also allows users to work with devices already familiar to them.

Trackballs may be preferred for reasons other than physical impairment. Trackballs need little space in which to operate (Figure 1b), unlike mice, which have large "desktop footprints" [6]. Trackballs can be embedded in consoles or keyboards, making them suitable for public terminals since they cannot be easily stolen. Trackballs offer rapid, fluid control and have been used in arcade games like *Centipede*.

CHI 2006, April 22-27, 2006, Montréal, Québec, Canada.

In this paper, we present a novel method for trackball text entry called *Trackball EdgeWrite*. This method relies not on precise pointing at on-screen targets, but on unistroke gestures that mimic the feel of writing Roman letters [25]. Instead of pointing, the method uses *crossing* [3], lowering the demand for accuracy. We evaluate the method in a single session with able-bodied users. Then we present results from an extended field study with a veteran trackball user with a spinal cord injury. A former on-screen keyboard user for 15 years, he now uses Trackball EdgeWrite instead.

RELATED WORK

Evaluations of Trackballs

Although there is little work on trackball text entry, there has been a great deal of work on trackball pointing. An early study by Epps et al. [7] compared six pointing devices, including a 4 cm trackball, in target acquisition tasks. They found that the mouse and trackball were significantly faster than the other devices, but were not significantly different from each other. A follow-up study by Sperling and Tullis [22] found that the mouse was faster for selection, dragging, and tracing even among trackball users. Accuracy differences were not significant for selection and dragging, but were significant for tracing, showing the trackball to be less accurate than the mouse. From these results, we can conclude that our trackball text entry method should not require precise positioning or tracing along a path. For example, we would not want a method that requires smooth freeform gestures like those in handwriting or most unistroke methods (e.g. [10]).

Accot and Zhai formalized tracing performance with their Steering Law [1]. Their study of five input devices [2] shows that trackballs are comparable to touchpads—but both are worse than mice—and that trackballs perform best relative to other devices for short straight-line trajectories less than 250 pixels. They note that the performance of trackballs and touchpads suffers when the device must be "clutched" for travel over long distances. From this we can conclude that long or curved movements should be avoided.

Further comparisons of pointing devices by MacKenzie et al. [15] showed trackballs to be slower than mice and styli in pointing and dragging, and less accurate for dragging. They note that dragging is particularly difficult with a trackball because of the confluence of the thumb and finger muscles. In a later study, MacKenzie et al. [14] added that trackballs often move accidentally when performing a click inside a target. Such slips can be particularly troubling for users with motor impairments [23]. Thus, our method should not rely on buttons and not require dragging or clicking. It should therefore be button-free.

After this brief review, it may seem that trackballs are unlikely candidates for gestural text entry. However, this conclusion ignores the strengths of trackballs that make them preferable for many motor-impaired users. Trackballs require little if any forearm or wrist movement, little strength, little desk space, and offer rapid, fluid control. Although selecting, dragging, and steering are not strengths of trackballs compared to mice, other activities such as goal crossing may be better suited.

Goal Crossing

Unlike pointing, which requires a cursor to enter a target and remain stationary long enough to select it, *crossing* requires only that a goal line be passed, like a football player scoring a touchdown. This can lessen the demands for target acquisition compared to traditional pointing.

Accot and Zhai [1,3] modeled goal crossing and showed it to follow a formulation of Fitts' Law [8]. Specifically, they showed that crossing goals of width *W* separated by a distance *A* (Figure 2) follows the Fitts' formulation (Equation 1). However, the regression coefficients *a* and *b* are different for crossing than for pointing [3].

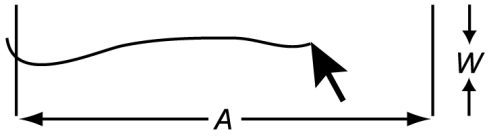

Figure 2. Crossing follows Fitts' law but with the *W* constraint orthogonal to the movement trajectory, rather than collinear.

$$T = a + b \cdot \log_2\left(\frac{A}{W} + 1\right) \quad (1)$$

Accot and Zhai later elaborated by comparing different types of pointing and crossing [3]. They found that the fastest arrangement for reciprocal tasks was "continuous orthogonal goal crossing," where the pointing device, in their case a stylus, was continually held on the surface and the goals were perpendicular to the movement trajectory (Figure 2). They speculated that goals could rotate to always remain orthogonal to the cursor and thus offer the maximum target width (Figure 3a). An extreme form of this idea is a cursor placed inside a circle, where the circumference is the goal (Figure 3b). This insight drives our design for Trackball EdgeWrite, and the crossing model provides for its analysis.

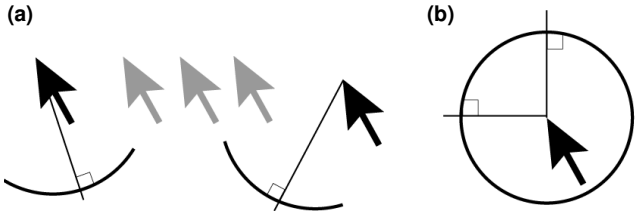

Figure 3. (a) Accot and Zhai [1] speculate that crossing goals could rotate to always remain orthogonal to the cursor, thereby offering maximum target width. **(b)** An extreme form of this idea is a cursor in the center of a circle, where the circumference itself is the goal.

Related Text Entry Methods

There are numerous text entry methods for mobile devices and computer access. Space precludes a full review. Readers are directed to prior work on EdgeWrite [29] and MacKenzie and Soukoreff's text entry overview [16].

To our knowledge there has not been any text entry method developed specifically for trackballs. But two methods of note work with them. One is MDITIM [11], a stroke-based method that is usable on different devices, including trackballs. In a study of MDITIM on different devices, the trackball was measured at about 6.5 wpm with 7.2% errors after ten 30-minute sessions with a stylus. The test subjects knew the MDITIM alphabet but were not necessarily proficient with a trackball. One drawback of MDITIM is that all letter forms are comprised of movements in only the four cardinal directions. Therefore, MDITIM letters do not generally resemble their Roman counterparts, making guessability and learnability difficult.

Dasher [24] is another method that works with trackballs. Dasher works by having the user select from an ever-expanding array of dynamic rectangular regions that represent letters. The regions' sizes, and therefore the ease of their selection, depend on the linguistic properties of the text being entered. Like Trackball EdgeWrite, Dasher requires no clicking or dragging. Although Dasher can achieve rapid entry rates (~30 wpm), a common sentiment among novices is that it is overwhelming, as it is in constant visual flux. Also, it is unlikely that Dasher could be done purely by feel without any graphical depiction [24]. Dasher can be used as an assistive technology with a head- or eye-tracker, since any pointing device will work with it.

A common method of trackball text entry is with an on-screen keyboard. On-screen keyboards have been designed [5] and studied [19] in computer access for some time. Although word prediction can help raise speeds, it can also have the opposite effect, depending on the severity and nature of the impairment [12]. Even if performance is enhanced, on-screen keyboards remain a visually taxing and tedious method of text entry. Our own field test subject remarked that he would prefer a gestural method that he could use "by feel" over an on-screen keyboard, even if the keyboard were faster. Fortunately, this subject was faster with Trackball EdgeWrite than his own on-screen keyboard, so he did not have to suffer this tradeoff.

TRACKBALL EDGEWRITE — FUNDAMENTAL DESIGN

Background

EdgeWrite was originally developed for use with a stylus on a PDA by people with tremor [29]. That version placed a plastic template over a Palm PDA's input area, thereby providing a square hole in which all strokes could be made (Figure 4). EdgeWrite's mnemonic, Roman-like unistroke alphabet enables users to write by moving the stylus along the physical edges and into the corners of the plastic square. To tolerate wiggle, recognition was based on a stroke's order of corner-hits, not on its overall path of motion. The physical stability provided by this design increased accuracy among motor-impaired users by 2-4 times over Graffiti [29].

Figure 4. An EdgeWrite template over the text input area of a Palm PDA. Physical edges and corners provide stability for the stylus.

EdgeWrite's alphabet was designed through a guessability study in which participants devised intuitive gestures for each letter [25]. Thus, EdgeWrite gestures are not short and efficient like the original Unistrokes [10], but are slightly longer and more reminiscent of Roman forms (Figure 5). This has made EdgeWrite highly guessable and quickly learnable [25], an important feature for motor-impaired users who often cannot endure lengthy practice sessions [28].

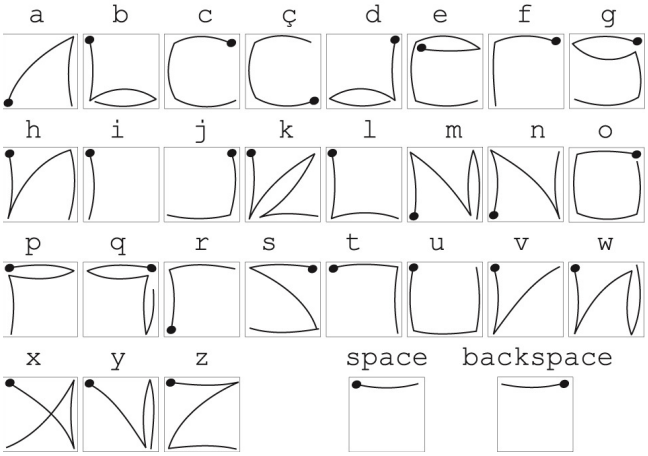

Figure 5. EdgeWrite's primary letter forms. Alternate forms exist for most letters (not shown). The bowing of line segments is for illustrative purposes only—all motion is in straight lines. EdgeWrite also defines strokes for punctuation, accents, and extended characters. For a complete chart, see http://www.edgewrite.com.

EdgeWrite's method of stroke recognition—detecting the order of corner-hits—has lent itself to versions for touchpads, joysticks, buttons, and other devices [26, 27, 28]. However, previous adaptations have relied exclusively on *absolute positioning* in performing input, i.e., the physical position of a stylus, finger, or joystick within a physical square. Trackball EdgeWrite is the first version to use *relative positioning*, since trackballs themselves express no absolute position or rotation, only change in rotation [6]. In relative positioning, the mouse cursor itself becomes the locus of input, and issues arise that are different than in previous versions. In cursor-based relative positioning, physical edges are no longer possible. Instead, the accuracy benefits formerly provided by physical edges are obtained with crossing [3], as detailed in the next section.

Crossing to Stroke

With Trackball EdgeWrite, one writes by making short "pulses" toward intended corners. When these pulses cause the mouse cursor to cross the circumference of a circle maintained about it, the resultant angle indicates the next corner. Thus, the cursor does not actually travel among corners to acquire them as targets, but crosses a radius and *snaps* to the next corner instantly. Figure 6 depicts this process for writing the letter "z".

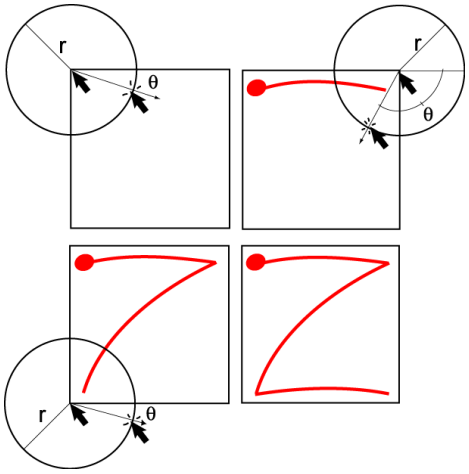

Figure 6. The trackball cursor moves a distance *r* at an angle *θ* from the top-left corner of the EdgeWrite square. The angle determines that the next corner is the top-right, and the first stroke of the letter "z" is drawn. Having snapped to the top-right corner, the cursor now moves diagonally at an angle that indicates the bottom-left corner, and the second stroke is drawn. Note that for the third stroke, the cursor moves towards the *outside* of the EdgeWrite square, but at an angle that still indicates the bottom-right corner.

In essence, a series of short crossing tasks forms a letter. But clearly, much depends on how the circle is partitioned and how θ is interpreted. For instance, consider the move across the bottom of the "z" in Figure 6. At what angle θ should the cursor instead move diagonally to the top-right?

Figure 7 shows the next-corner outcomes for different departure angles from the bottom-left corner of the EdgeWrite square. The same scheme can be extrapolated to the other three corners of the EdgeWrite square.

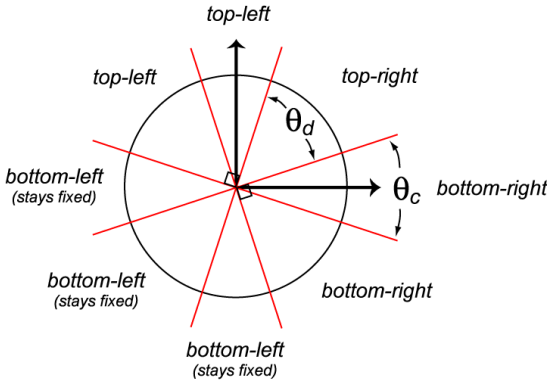

Figure 7. Next-corner outcomes for different angles of departure from the bottom-left corner of the EdgeWrite square. θ_d is the diagonal angle size and θ_c is the cardinal angle size.

Three features of the design in Figure 7 are important. First, any movement left, diagonal down-and-left, or down keeps the cursor fixed at the extreme bottom-left corner. More generally, if the cursor moves at an angle that, were it to cross the circumference it would remain in the same corner, it is held fixed in the center of the circle. This will always hold for $(180 - \theta_d)°$ of the circle, where θ_d is the angular amount allotted to each diagonal. This "pinning" to the circle's center ensures that one remains accurate if one is slow to change directions after entering a new corner.

Second, the value in having ample cardinal angles (θ_c) is that users do not always move perfectly to the left, right, up or down. Larger values of θ_c create more room *inside* the EdgeWrite square to move in the cardinal directions. However, adding to θ_c detracts from the diagonals θ_d because $\theta_c + \theta_d = 90°$.

Third, although increasing θ_c gives more room inside the square to move in the cardinal directions, one *always* has 90° with which to move along an edge, regardless of θ_c. That's because one always has the angles to the *outside* of the EdgeWrite square, as in the bottom-left image in Figure 6. So although θ_d and θ_c are tradeoffs, a total of 90° remains for moving along an edge. In practice, however, we have found that some room in θ_c acts as a "cardinal error band," giving more room to move inside the EdgeWrite square.

It is important to emphasize that despite these underlying mechanics, one neither sees a visual cursor nor aims for particular angles when writing with Trackball EdgeWrite. Instead, one simply pulses the ball towards intended corners, creating the feeling of fluid writing.

Two Schemes for Determining the First Corner

The example in Figure 6 assumes that we start in the top-left corner. But of course, not all EdgeWrite letters start there. This begs the question of how we begin a letter, since unlike a stylus landing on a PDA or a finger landing on a touchpad, the trackball cursor is persistent and cannot simply "appear" in the starting corner.

We implemented two competing schemes for determining the first corner. The first scheme is to have the user pulse the trackball to the initial corner as if he was starting from the center of the EdgeWrite square. Thus, to resemble the top-left image in Figure 6, one would first pulse the trackball diagonally to the top-left. Although this scheme adds an extra movement to the start of every letter, the accuracy demands are low because the user has a full 90° with which to indicate each starting corner.

In the second scheme, one *assumes* one starts in the appropriate corner, and the software disambiguates that corner as the gesture unfolds. For example, in making a "z", one would first move to the right (Figure 8). At this point, one may have intended to move along either the top or bottom edge. Both possibilities are entertained until the next move is diagonally down-and-left, at which point the ambiguity is resolved. Gestures that occur along a single

edge, like "i", space, and backspace, never resolve into unambiguous strokes. But such gestures can be defined on both sides of the square, so the ambiguity can be made irrelevant.

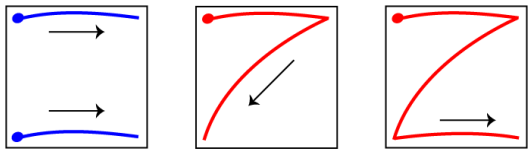

Figure 8. When writing a "z" in the second scheme, one first moves to the right, causing two candidate strokes to appear. After the diagonal stroke, the gesture is no longer ambiguous, and the bottom-edge candidate is discarded. A third stroke finishes the "z".

Although the second scheme requires one less pulse per letter than the first scheme, it proved slower and more difficult in practice because the initial stroke has eight possible outcomes, not just four as in the first scheme. Even when the angular regions of the circle were allocated proportional to the probability of beginning a letter in that region, the second scheme proved too difficult for our subjects to perform accurately. Furthermore, our theoretical analysis of each scheme actually showed the first scheme to be 3 wpm *faster* despite the additional stroke required for each letter because of the high accuracy required for the second scheme's initial stroke. Thus, the first scheme was preferred. This was an unexpected finding, as we had initially assumed the second scheme would be about 30% faster. That theoretical and empirical results proved otherwise shows the importance of basing designs on quantitative measures instead of intuition.

Theoretical Analysis

We can predict the time it takes to make a letter using Accot and Zhai's [1,3] discovery that crossing follows the Fitts formulation in Equation 1.

For diagonal movement, the size of the crossing goal is:

$$W_d = \frac{\theta_d}{360} \cdot 2 \cdot \pi \cdot r \tag{2}$$

The index of difficulty for diagonal movement is therefore:

$$ID_d = \log_2\left(\frac{r}{\frac{\theta_d}{360} \cdot 2\pi r} + 1\right) \tag{3}$$

Thus, our time for diagonal movements is predicted by:

$$T_d = a + b \cdot \log_2\left(\frac{180}{\theta_d \cdot \pi} + 1\right) \tag{4}$$

For movement in the cardinal directions, recall that we have a constant target size of 90°. Thus,

$$W_c = \frac{90}{360} \cdot 2\pi r \tag{5}$$

So, our index of difficulty is:

$$ID_c = \log_2\left(\frac{r}{0.5\pi r} + 1\right) \tag{6}$$

Our movement time along cardinal edges is thus given by:

$$T_c = a + b \cdot 0.710719 \tag{7}$$

For the pulse from the center to the first corner, we have four target angles each of 90°. Thus, our movement time T_f for the pulse to the first corner is the same as T_c:

$$T_f = T_c = a + b \cdot 0.710719 \tag{8}$$

Using Equations 4, 7, and 8, we can compute the theoretical movement time for each EdgeWrite character shown in Figure 5. For example, using a typical 65° for diagonal angles θ_d and 150 ms for segmentation timeout τ, the time in milliseconds to enter the letter "z" is given by:

$$\begin{aligned} T_{"z"} &= T_f + T_c + T_d + T_c + \tau \\ &= 3 \cdot T_c + T_d + \tau \\ &= 4a + 3.04402\ b + 150 \end{aligned} \tag{9}$$

To compute the time of the "average character," we weight each character's time by its linguistic frequency.

$$T_{avg} = \sum_{i \in C} T_i \cdot F_i \tag{10}$$

In Equation 10, i is a character in character set C, T_i is the time (ms) for character i, and F_i is the linguistic frequency for character i. For simplicity, we use Figure 5 for C but omit "ç" and backspace. We do, of course, include space.

Finally, using Equation 10, we approximate the theoretical speed for the method. (The numerator is ms/min.)

$$wpm = \frac{60{,}000}{5 \cdot T_{avg}} \tag{11}$$

Obviously, *wpm* is dependent upon our choice of a and b—our Fitts' regression coefficients. To our knowledge, no studies have elicited these coefficients for continuous reciprocal goal crossing with a trackball. However, Accot and Zhai have done so for a stylus [3]. They have also done so for a stylus and a trackball for non-reciprocal straight-tunnel steering [2]. Extrapolating from these studies, we can obtain estimates for a and b for continuous reciprocal goal crossing with a trackball. Using a typical θ_d = 65° and τ = 150 ms, Equation 11 yields the following:

wpm = 23.1 $\qquad$ a = -363.0, b = 642.1

Note that this rate represents *perfect* entry, i.e., no errors, no error correction, and no hesitation between letters other than the time τ required for segmentation. Also, trackballs differ significantly in their sizes, friction, gain settings, etc. Thus, our estimate is only a ballpark measure. However, it is useful for comparing different stroking schemes, such as the two first-corner schemes discussed in the previous section. An item for future work is to establish a and b coefficients for reciprocal goal crossing with a trackball.

TRACKBALL EDGEWRITE — DESIGN REFINEMENTS

Non-recognition Retry

With most gestural text entry methods, there is no way to predict what a recognizer will produce until a character appears, at which point it is too late to save an erroneous gesture. With EdgeWrite, since all characters are defined by their order of corner-hits, users often realize they've made a mistake before a gesture is complete. Rather than wait for the recognizer to produce an erroneous letter and then have to erase it, our design allows users to *restart letters mid-stroke without segmenting*. When given the full "garbled" gesture to recognize, EdgeWrite trims corners from the start of the corner sequence until it finds a sequence it recognizes. We call this "non-recognition retry" (Figure 9).

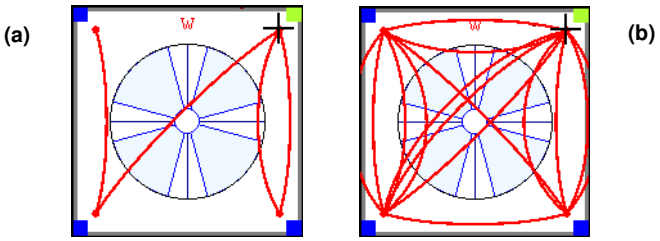

Figure 9. (a) A clean "w" and **(b)** a jumbled "w". The gesture at right was begun erroneously but corrected mid-stroke to finish as a "w". Non-recognition retry discovers the "w" at the end of the stroke.

Non-recognition retry is possible in EdgeWrite because gestures are effectively tokenized into corners as they are written. It would be difficult to reliably apply non-recognition retry to freeform gestures, since one would have to perform some preliminary recognition to determine where to trim the point queue in the first place.

Slip Detection, Multiple Hypotheses, and Digraphs

When writing with Trackball EdgeWrite, occasionally one slips through an unwanted corner when trying to make a diagonal. To mitigate this problem, the system includes a *slip detector* that adapts to the speed of the writer. The slip detector tracks the average inter-corner time ψ for the last n corners. If after being in a corner c_1, a stroke is made abruptly at p percent of ψ from a corner c_2 to c_3, and c_1 and c_3 lie on a diagonal, then the stroke from c_1 to c_2 is removed and a connection is made directly from c_1 to c_3. That is, c_2 is deemed to have been "slipped through" on the way from c_1 to c_3. Through empirical testing we found that n = 16 and p = 37.5% work well. Figure 10 depicts this scenario.

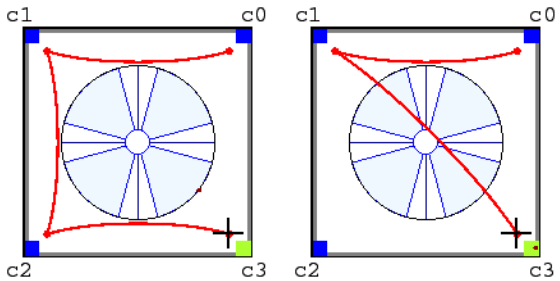

Figure 10. In writing an "s", the user slips through c_2 while trying to move diagonally from c_1 to c_3. The slip is detected and the stroke from c_1 to c_2 is replaced with a connection directly from c_1 to c_3.

However, the slip detector is "humble." It knows it may not always be right. Therefore, upon detecting a slip, it keeps two corner sequences: one with the slip removed, and the other with the slip retained. In doing this for each slip within a gesture, it builds a binary tree as shown in Figure 11, with tiers in the tree representing perceived slips.

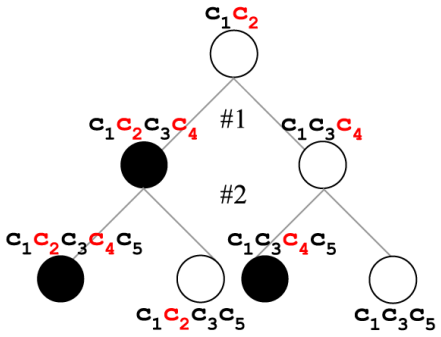

Figure 11. The slip detector retains two hypotheses for every detected slip, namely that the slip did and did not occur. At slip #1, the corner c_2 was passed through abruptly and is retained (left) and removed (right). At slip #2, the corner c_4 was passed through too quickly and is retained (left) and removed (right).

Upon segmentation, all leaves in the tree are recognized. Each recognition result is then checked for its likelihood given the preceding character using the digraph lists from [4]. The result with the greatest likelihood is produced as the recognized character. For example, "pu" is a much more common English digraph than "pv". Thus, if a "u" is corrected to a "v" due to a perceived slip, the "u" will still be produced if the letter before it is a "p".

Interaction Design

Trackball EdgeWrite is the first EdgeWrite version to utilize the mouse cursor. Two key issues arise when building an application of this sort: how to switch between mousing and writing, and how to segment between letters.

Capture and Release

Trackball EdgeWrite is designed to run quietly in the background until it is needed for text entry. When the mouse is being used for mousing, it is said to be "released." When it is being used for writing, it is said to be "captured." The mouse can be captured by clicking a dedicated trackball button (a "hot button"), pressing a dedicated keyboard key (a "hot key"), or by dwelling in a corner of the desktop (a "hot corner"). Hot buttons, keys, and corners are configurable in the application's preferences dialog.

When captured, the mouse cursor vanishes and the EdgeWrite window appears near the text cursor (Figure 12). To release the cursor, the user can click a trackball button or perform a dedicated release stroke. Then the EdgeWrite window disappears and the mouse cursor is restored to its previous location. Thus, Trackball EdgeWrite never requires that it be navigated to, unlike an on-screen keyboard. Instead, it comes to the user when it is summoned.

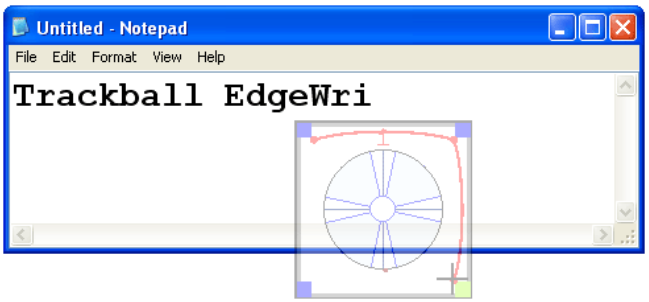

Figure 12. Trackball EdgeWrite is being used to enter text into Notepad. Although mouse events are being handled by EdgeWrite, the input focus appropriately remains on the Notepad window.

Segmentation

Letters are segmented when the underlying mouse cursor stops moving for a preset amount of time. (This time was modeled as τ in our theoretical analysis, above.) Timeout values range from 100 ms for experts to 750 ms for novices. This simple scheme works reliably. A beginner can still "stop and think" while making a gesture by slowly rolling the trackball toward the corner they're already in, as this will not move them to a new corner (recall Figure 7).

Implementation

Trackball EdgeWrite is implemented in C# using DirectInput to capture mouse events in the background. Recognized characters are sent through the low-level keyboard input stream. Trackball EdgeWrite works with any pointing device, but is best suited for devices without absolute position, e.g., trackballs and isometric joysticks. EdgeWrite allows letters, numbers, punctuation, diacritical marks, and extended characters. It can also act as a scroll ring for rapid cursor movement. Thus, with a miniature trackball or an isometric joystick, EdgeWrite provides full text entry in a small space. It can be downloaded for free at http://www.edgewrite.com/dev.html.

LAB STUDY WITH ABLE-BODIED USERS

For comparisons, we conducted a study of Trackball EdgeWrite with able-bodied subjects who knew the EdgeWrite alphabet but who were not trackball users. Their knowledge of the EdgeWrite alphabet came from a previous study with a different EdgeWrite version. This allowed us to isolate performance with the trackball apart from learning the alphabet. The study was not intended to compare two opposing methods, but to provide an estimate of the performance of Trackball EdgeWrite for novices.

Method

Three subjects ages 27-33 took part in the study. All were daily computer users who owned a desktop mouse. None had ever owned or used a trackball. They were paid $15.

Text phrases from [17] were shown in Courier 20pt on an IBM A21p laptop set to 1280×1024 resolution. The phrases were transcribed with Trackball EdgeWrite using a Kensington Expert Mouse trackball (Figure 1). The pointer speed was set to 40% of maximum on the mouse control panel. Backspace was supported and subjects were not required to stay synchronized with the presented text. They were told to write quickly and accurately, and to balance speed and accuracy without favoring one over the other.

Subjects practiced for 45 minutes with Trackball EdgeWrite before transcribing 15 test phrases. During practice, the three subjects transcribed 58, 26, and 40 practice phrases, respectively. Although they knew the EdgeWrite letters, they had a low level of comfort with the trackball, feeling that it would take more time to become comfortable with it.

Able-bodied Results

The 15 test phrases were analyzed according to the measures in [20]. No analyses of variance were performed since only one method was under investigation. Subjects' speeds and error rates are shown in Figures 13 and 14. As an additional point of comparison, the first author's performance is shown at the right of each graph. For the three subjects, the average entry rate was 9.82 wpm. Their average respective uncorrected, corrected, and total error rates as defined in [20] were 0.59%, 3.31%, and 3.90%.

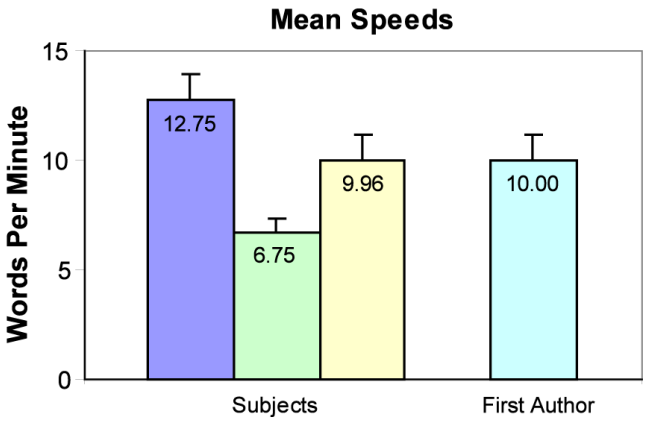

Figure 13. Average speeds (wpm) for three able-bodied subjects using Trackball EdgeWrite. Subjects knew the alphabet but had never used trackballs.

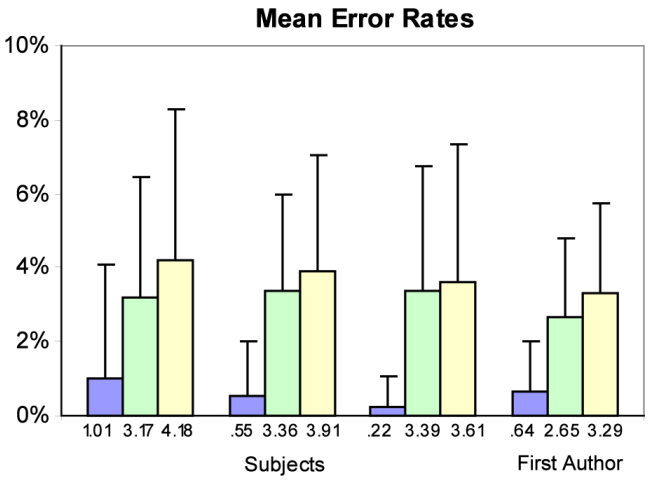

Figure 14. Average error rates for each subject. From left to right, error rates within each subject are uncorrected, corrected, and total [20]. Total error rates were quite low (<4%) for a gestural technique using an unfamiliar device.

FIELD STUDY WITH A MOTOR-IMPAIRED USER

Although 10 wpm is not particularly fast, a trackball user with motor impairments resigned to using an on-screen keyboard at 4-5 wpm would probably welcome that speed. In view of this, we improved Trackball EdgeWrite over the course of eight participatory design sessions conducted from May to October 2005 with a veteran trackball user who we will call "Jim." Jim has a spinal cord injury that reduces the dexterity in his arms, hands, and fingers such that he cannot use a conventional keyboard or mouse. For 15 years he has relied upon trackballs and on-screen keyboards for computer access. His favorite trackball is the Stingray from CoStar Corporation (Figure 15).

Figure 15. Jim's favorite trackball is the Stingray. He prefers this trackball because he can press its wide left button with the palm of his left hand while his left thumb rolls the ball to drag.

Jim is now 46 years old and is still an avid trackball user. But he is a reluctant on-screen keyboard user. For extended periods of text entry, Jim uses Dragon Naturally Speaking, but he nonetheless frequently relies on an on-screen keyboard. He complains that his speech recognition often "acts up" and ceases to work well. Sometimes his voice is altered from medications or fatigue and his recognition rates drop. For short replies to emails, putting on a headset and microphone is an arduous task, so Jim uses an on-screen keyboard when he needs to enter a few words. Also, certain tasks don't work well for him with speech recognition, like naming files, entering email addresses, editing proper names, entering spreadsheet formulae, and filling out web forms. Thus, Jim's text entry solution has been a mixture of speech recognition and an on-screen keyboard using a trackball to dwell over letters. At one time he used Click-N-Type (http://www.lakefolks.org/cnt/), but when we met him he had switched to Microsoft's Accessibility Keyboard.

Jim has three main complaints about using an on-screen keyboard. First, the visual attention required is enormous, as he constantly must look from the keyboard to his document and back. Second, moving over keys to dwell requires that the dwell time be long enough to avoid unwanted keys but short enough to produce text quickly, a difficult balance to strike. He currently uses 0.5 seconds as the dwell time. Third, the tedium of making repeated keyboard selections is, according to Jim, "mind numbing." Although word prediction can help, Jim feels that word prediction slows him down as often as it speeds him up, a sentiment consistent with findings in the literature [12]. Plus, word prediction adds more items to visually scan.

Performance

To ensure that our design iterations were beneficial, we performed short checkpoint studies when meeting with Jim. During these studies, Jim would use the latest version of Trackball EdgeWrite to enter 5-7 test phrases from [17]. He would also enter the same number of test phrases with the Microsoft Accessibility Keyboard. Word prediction was not available in either method. Both methods were controlled by Jim's personal trackball (Figure 15), and neither method required any button presses. Jim's speed and accuracy results are shown in Figures 16-19 on the next page.

Note that the checkpoint studies were not evenly spaced in time, particularly in the jump from week 4 to week 10. During this 6 week hiatus, Jim did not use his computer. Thus, the data for week 10 can be viewed as retention results. Week 10 was the only week after week 1 in which the two methods were nearly equal in speed (Figure 16).

After week 1, Jim's speeds were consistently higher with EdgeWrite than with the on-screen keyboard (Figure 16). His average speed across all weeks with EdgeWrite was 5.28 wpm (σ=1.05). With the keyboard it was 4.60 wpm (0.94). A Wilcoxon sign test for matched pairs yields a significant result in favor of EdgeWrite (z=-15.0, p<.04).

The drop in performance during week 15 was due to a nagging tremor in Jim's hand. The drop in speed occurred about evenly for both methods (Figure 16). Accuracy results, on the other hand, show that both uncorrected and corrected errors were worse for the on-screen keyboard than for EdgeWrite (Figures 17-18), suggesting that EdgeWrite accuracy may be less affected by such tremors.

Jim's average uncorrected error rate was 1.18% (σ=1.47%) with EdgeWrite and 2.05% (2.62%) with the on-screen keyboard. A Wilcoxon sign test for uncorrected errors is not significant (z=5.5, p=0.31). His average corrected error rate was 10.62% (1.80%) with EdgeWrite and 4.49% (3.93%) with the keyboard. A Wilcoxon sign test for corrected errors does yield a significant result in favor of the on-screen keyboard (z=-17.0, p<.02).

Although corrected errors (Figure 18), which are any letters backspaced during entry, are higher with Trackball EdgeWrite, they are not necessarily damaging if the correction operation is efficient [20]. Of course corrected errors should be reduced, but the main tradeoff is between speed and *uncorrected* errors (Figure 17), which are errors left in the transcribed string. A method can be successful even if it quickly produces and repairs errors during entry if, in the end, it yields more accurate text in less time, which is the case here.

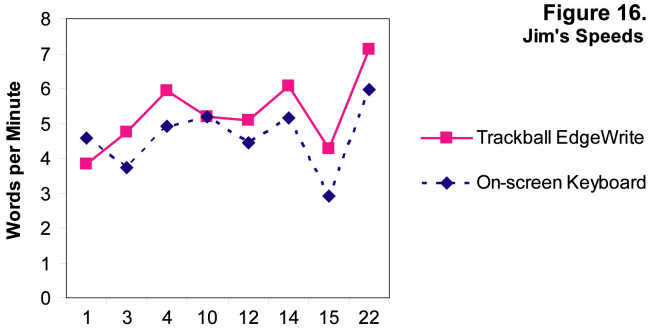

Figure 16.
Jim's Speeds

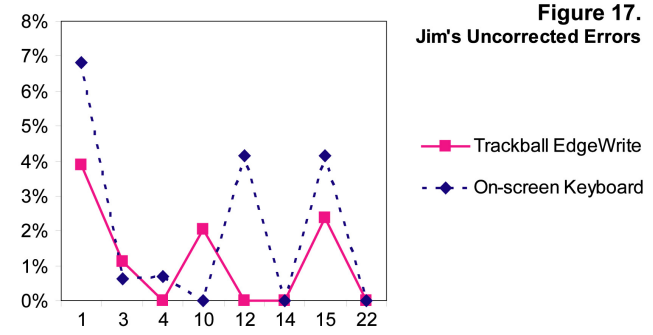

Figure 17.
Jim's Uncorrected Errors

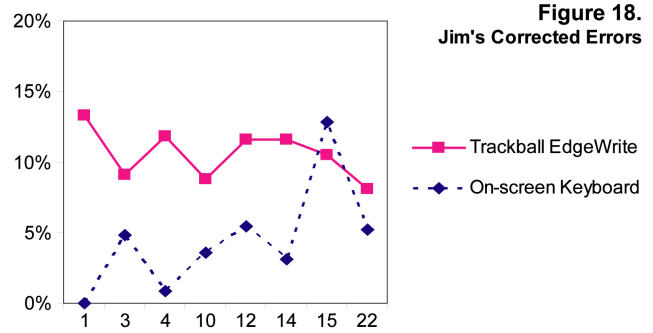

Figure 18.
Jim's Corrected Errors

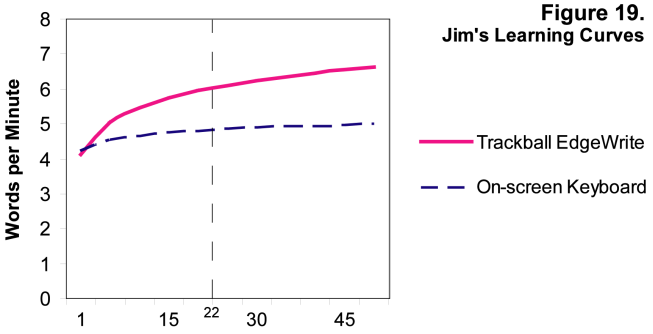

Figure 19.
Jim's Learning Curves

Figures 16-19. Jim's average speeds, uncorrected errors, corrected errors, and learning curves across weekly sessions.

Figure 19 shows Jim's speed curves modeled by the power law of learning and extrapolated to future sessions as was done in [18]. The regression equations are:

$$\textit{EdgeWrite}: \quad y = 4.1369x^{0.1208} \qquad R^2 = 0.3579$$
$$\textit{Keyboard}: \quad y = 4.2358x^{0.0423} \qquad R^2 = 0.0429$$

The R^2 values indicate that Trackball EdgeWrite is better modeled by a learning curve. This is probably because Jim was already an expert on-screen keyboard user and had little room for improvement.

Qualitative Results

In general, the time Jim spent using EdgeWrite each week varied according to his personal computer use and his text entry needs. With Jim's permission, Trackball EdgeWrite wrote log files on his computer whenever it was being used. These logs show that EdgeWrite was always left running in the system tray as of week 3, and was used intermittently over the course of most weeks. The total use per week varied from just a few minutes to a few hours. Jim's usage increased in the later sessions.

Jim no longer uses an on-screen keyboard. His main reasons for preferring Trackball EdgeWrite over an on-screen keyboard are, in his words:

- "The on-screen keyboard is so terribly boring. EdgeWrite is fun, like a video game. The on-screen keyboard is not fun. I don't care which is faster."
- "With EdgeWrite, you can keep your eyes on your document and just write as you would holding a pencil. I don't feel disabled when I'm using EdgeWrite."
- "The on-screen keyboard requires too much visual scanning and concentration. In EdgeWrite, if you know the letter, you can just bang it out by feel."

As Jim began working with Trackball EdgeWrite, he also began to use a Palm PDA for the first time. Previously, he could not enter text into a PDA because of the difficulty he had in holding a stylus and moving it accurately. Now he uses EdgeWrite with a stylus on his Palm PDA [29], and says he can take handwritten notes for the first time in years. A nice aspect of this is that Jim only had to learn the EdgeWrite alphabet once.

FUTURE WORK

A limitation of the current work is that it only used a single motor-impaired subject over multiple sessions. This was to promote rapid iteration of the design. Thus, future work includes a larger user study and a wider deployment for people with disabilities.

We also intend to add word completion to Trackball EdgeWrite via *in-stroke word completion*, which will allow users to complete words fluidly by the manner in which their letter gestures are finished. We also noted that Trackball EdgeWrite works well with isometric joysticks, and intend to apply it to mobile phones by placing a joystick on the front of the phone for use with a thumb, and on the back of the phone for use with an index finger.

CONCLUSION

In presenting Trackball EdgeWrite, we have shown how prior studies, theoretical models, and empirical results can combine to rigorously inform the design of new input techniques. The foundation of our design lies in *crossing* instead of pointing, and represents a new way to do gestural input. Although intuition was useful as a starting point, our empirical and theoretical results contradicted our intuition

when considering the two different first-corner schemes, indicating the value of quantitative evaluations over intuition. Our experience supports a rapid build-and-evaluate cycle with few subjects instead of a large but premature user study. Now that our design has been refined, a large user study is appropriate. Our results should encourage researchers to explore new ways in which common input devices and fundamental techniques can be used to improve computer access for users with disabilities.

Acknowledgements

The authors thank John SanGiovanni, Richard Simpson, John Kembel, Jeffrey Nichols, and especially "Jim." This work was supported in part by Microsoft, General Motors, and the National Science Foundation under grant UA-0308065. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

REFERENCES

1. Accot, J. and Zhai, S. (1997) Beyond Fitts' law: Models for trajectory-based HCI tasks. *Proc. CHI '97*. ACM Press, 295-302.
2. Accot, J. and Zhai, S. (1999) Performance evaluation of input devices in trajectory-based tasks: An application of the Steering Law. *Proc. CHI '99*. ACM Press, 466-472.
3. Accot, J. and Zhai, S. (2002) More than dotting the i's: Foundations for crossing-based interfaces. *Proc. CHI '02*. ACM Press, 73-80.
4. Bellman, T. and MacKenzie, I.S. (1998) A probabilistic character layout strategy for mobile text entry. *Proc. Graphics Interface '98*. CIPS, 168-176.
5. Bishop, J.B. and Myers, G.A. (1993) Development of an effective computer interface for persons with mobility impairment. *Proc. 15th Annual Conference on Engineering in Medicine and Biology*. IEEE Press, 1266-1267.
6. Card, S.K., Mackinlay, J.D. and Robertson, G. (1990) The design space of input devices. *Proc. CHI '90*. ACM Press, 117-124.
7. Epps, B.W., Snyder, H.L. and Muto, W.H. (1986) Comparison of six cursor devices on a target acquisition task. *Proc. Society for Information Display*. Society for Information Display, 302-305.
8. Fitts, P.M. (1954) The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47* (6), 381-391.
9. Fuhrer, C.S. and Fridie, S.E. (2001) There's a mouse out there for everyone. *Proc. CSUN '01*. California State University Northridge.
10. Goldberg, D. and Richardson, C. (1993) Touch-typing with a stylus. *Proc. CHI '93*. ACM Press, 80-87.
11. Isokoski, P. and Raisamo, R. (2000) Device independent text input: A rationale and an example. *Proc. AVI '00*. ACM Press, 76-83.
12. Koester, H.H. and Levine, S.P. (1994) Validation of a keystroke-level model for a text entry system used by people with disabilities. *Proc. ASSETS '94*. ACM Press, 115-122.
13. Koester, H.H. and Levine, S.P. (2000) User performance with continuous speech recognition systems. *Proc. RESNA '00*. RESNA Press, 549-554.
14. MacKenzie, I.S., Kauppinen, T. and Silfverberg, M. (2001) Accuracy measures for evaluating computer pointing devices. *Proc. CHI '01*. ACM Press, 9-16.
15. MacKenzie, I.S., Sellen, A. and Buxton, W. (1991) A comparison of input devices in elemental pointing and dragging tasks. *Proc. CHI '91*. ACM Press, 161-166.
16. MacKenzie, I.S. and Soukoreff, R.W. (2002) Text entry for mobile computing: Models and methods, theory and practice. *Human Computer Interaction 17* (2), 147-198.
17. MacKenzie, I.S. and Soukoreff, R.W. (2003) Phrase sets for evaluating text entry techniques. *Ext. Abstracts CHI '03*. ACM Press, 754-755.
18. MacKenzie, I.S. and Zhang, S.X. (1999) The design and evaluation of a high-performance soft keyboard. *Proc. CHI '99*. ACM Press, 25-31.
19. Shein, F., Hamann, G., Brownlow, N., Treviranus, J., Milner, M. and Parnes, P. (1991) WiViK: A visual keyboard for Windows 3.0. *Proc. RESNA '91*. RESNA Press, 160-162.
20. Soukoreff, R.W. and MacKenzie, I.S. (2003) Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric. *Proc. CHI '03*. ACM Press, 113-120.
21. Spaeth, D.M., Jones, D.K. and Cooper, R.A. (1998) Universal control interface for people with disabilities. *Saudi Journal of Disability and Rehabilitation 4* (3), 207-214.
22. Sperling, B.B. and Tullis, T.S. (1988) Are you a better "mouser" or "trackballer"? A comparison of cursor-positioning performance. *SIGCHI Bulletin 19* (3), 77-81.
23. Trewin, S. and Pain, H. (1999) Keyboard and mouse errors due to motor disabilities. *International Journal of Human-Computer Studies 50* (2), 109-144.
24. Ward, D.J., Blackwell, A.F. and MacKay, D.J.C. (2000) Dasher — A data entry interface using continuous gestures and language models. *Proc. UIST '00*. ACM Press, 129-137.
25. Wobbrock, J.O., Aung, H.H., Rothrock, B. and Myers, B.A. (2005) Maximizing the guessability of symbolic input. *Ext. Abstracts CHI '05*. ACM Press, 1869-1872.
26. Wobbrock, J.O. and Myers, B.A. (2005) Gestural text entry on multiple devices. *Proc. ASSETS '05*. ACM Press, 184-185.
27. Wobbrock, J.O., Myers, B.A. and Aung, H.H. (2004) Writing with a joystick: A comparison of date stamp, selection keyboard, and EdgeWrite. *Proc. Graphics Interface '04*. CHCCS, 1-8.
28. Wobbrock, J.O., Myers, B.A., Aung, H.H. and LoPresti, E.F. (2004) Text entry from power wheelchairs: EdgeWrite for joysticks and touchpads. *Proc. ASSETS '04*. ACM Press, 110-117.
29. Wobbrock, J.O., Myers, B.A. and Kembel, J.A. (2003) EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion. *Proc. UIST '03*. ACM Press, 61-70.