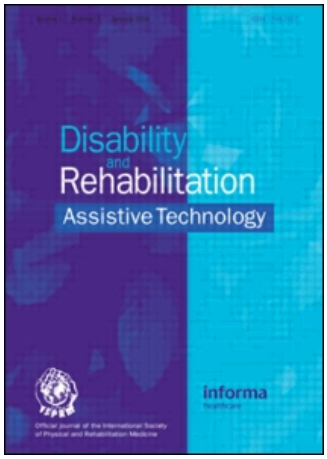

Disability and Rehabilitation: Assistive Technology
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t741771157

Enabling devices, empowering people: The design and evaluation of **Trackball EdgeWrite**

Jacob O. Wobbrock [a]; Brad A. Myers [b]
[a] University of Washington, Washington, USA
[b] Carnegie Mellon University, Pennsylvania, USA

First Published on: 13 June 2007
To cite this Article: Wobbrock, Jacob O. and Myers, Brad A. (2007) 'Enabling devices, empowering people: The design and evaluation of **Trackball EdgeWrite**', Disability and Rehabilitation: Assistive Technology, 3:1, 35 - 56
To link to this article: DOI: 10.1080/17483100701409227
URL: http://dx.doi.org/10.1080/17483100701409227

Disability and Rehabilitation: Assistive Technology, January/March 2008; 3(1 – 2): 35 – 56

Enabling devices, empowering people: The design and evaluation of *Trackball EdgeWrite*

JACOB O. WOBBROCK[1] & BRAD A. MYERS[2]

[1]*University of Washington, Washington, USA, and* [2]*Carnegie Mellon University, Pennsylvania, USA*

Abstract

Purpose. To describe the research and development that led to *Trackball EdgeWrite*, a gestural text entry method that improves desktop input for some people with motor impairments. To compare the character-level version of this technique with a new word-level version. Further, to compare the technique with competitor techniques that use on-screen keyboards.

Method. A rapid and iterative design-and-test approach was used to generate working prototypes and elicit quantitative and qualitative feedback from a veteran trackball user. In addition, theoretical modelling based on the Steering law was used to compare competing designs.

Results. One result is a refined software artifact, *Trackball EdgeWrite*, which represents the outcome of this investigation. A theoretical result shows the speed benefit of word-level stroking compared to character-level stroking, which resulted in a 45.0% improvement. Empirical results of a trackball user with a spinal cord injury indicate a peak performance of 8.25 wpm with the character-level version of *Trackball EdgeWrite* and 12.09 wpm with the word-level version, a 46.5% improvement. Log file analysis of extended real-world text entry shows stroke savings of 43.9% with the word-level version. Both versions of *Trackball EdgeWrite* were better than on-screen keyboards, particularly regarding user preferences. Follow-up correspondence shows that the veteran trackball user with a spinal cord injury still uses *Trackball EdgeWrite* on a daily basis 2 years after his initial exposure to the software.

Conclusions. *Trackball EdgeWrite* is a successful new method for desktop text entry and may have further implications for able-bodied users of mobile technologies. Theoretical modelling is useful in combination with empirical testing to explore design alternatives. Single-user lab and field studies can be useful for driving a rapid iterative cycle of innovation and development.

Keywords: *Text entry, text input, gestures, unistrokes, area pointing, goal crossing, word prediction, word completion, word-level stroking*

Introduction

Trackballs are the preferred pointing device for numerous computer users, particularly for many people with some form of motor impairment [1,2]. For people with low strength, poor coordination, wrist pain or limited ranges of motion, rolling a trackball is often easier than shuttling a mouse across the surface of a desk. Trackballs' accessible properties include that they do not require the wrist or forearm to be elevated; they do not occupy much physical space, making them suitable for placement in a person's lap or on a wheelchair tray; they are easy to manipulate, as rolling a trackball requires relatively little strength; if clutching[1] is necessary, one must only lift one's finger or hand, not the device itself (as with a mouse); and trackballs are simple, cheap, robust and readily available, factors that when absent are known to be barriers to adoption [3 – 5].

Not surprisingly, many people who prefer trackballs due to motor impairments also cannot type on a conventional physical keyboard. For these people, a text entry solution besides typing is required. Using a trackball for text entry may reduce physical movement among devices and the need for multiple devices to be within reach [6,7]. Thus, a common solution for these people is to use a trackball with an on-screen keyboard and to click or hover/dwell on the virtual keys.

Although on-screen keyboards are easy to learn, using on-screen keyboards for trackball text entry has many problems. For starters, on-screen keyboards

Correspondence: Jacob O. Wobbrock, The Information School, University of Washington, Seattle, WA 98195-2840, USA.

ISSN 1748-3107 print/ISSN 1748-3115 online © 2008 Informa UK Ltd.
DOI: 10.1080/17483100701409227

exacerbate mouse travel to and from a document, particularly for document editing. On-screen keyboards also introduce a second focus-of-attention, preventing a user's eyes from remaining wholly on his or her document [8,9]. Furthermore, on-screen keyboards require repeated target acquisitions for which trackballs are not well suited [10,11]. They are also visually fatiguing, equivalent to typing in a 'hunt-and-peck' fashion. Finally, on-screen keyboards consume precious screen real estate, reducing one's visual workspace and increasing the need for window management. Note that although word prediction and completion may increase the speed of on-screen keyboards, word prediction does not solve the above problems [8]. What is needed is a trackball text entry method that does not rely on on-screen keyboards and therefore avoids these drawbacks.

Trackballs are also relevant beyond the domain of assistive technologies. Trackballs come in many shapes and sizes (Figure 1) and may be preferred to conventional mice for reasons other than physical impairment. For instance, trackballs need little space in which to operate, unlike mice, which have large 'desktop footprints' [12]. Trackballs can be embedded in consoles or keyboards, making them suitable for public terminals (e.g. in libraries) since they cannot be easily lost or stolen. Trackballs also offer rapid, fluid control and have been used in arcade games like *Centipede*. Finally, trackballs can be made fairly small, making them suitable as thumb-controlled devices for mobile computers. Thus, the work presented in this article, although targeted at trackball users with motor impairments, has implications beyond the domain of assistive technology due to the various domains in which trackballs are used.

This article describes a new technology developed by the authors which provides a *gestural* means of writing with a trackball. This technology, called *Trackball EdgeWrite* (Figure 2), allows users to write 'by feel' using gestures rather than 'by sight' using on-screen keyboards. The result is a faster, less tedious method of trackball text entry for people who may already use trackballs but cannot touch-type on a physical keyboard. This includes people with repetitive stress injuries, spinal cord injuries, arthritis and some neuromuscular disorders.

After a brief treatment of related work, the iterative design of *Trackball EdgeWrite* is presented. This is followed by a theoretical model of expert writing speed. Next, results are presented from an extended field study with a veteran trackball user with a spinal cord injury. Results are shown for both the character-level and word-level versions of *Trackball EdgeWrite*. After 15 years of using a trackball with an on-screen keyboard, the veteran trackball user has switched to *Trackball EdgeWrite* for his everyday text entry needs. Follow-up correspondence shows that he still uses *Trackball EdgeWrite* on a daily basis 2 years after his initial exposure.

Related work

Trackballs as pointing devices

Previous work with trackballs has almost exclusively regarded them as pointing devices. An early study by Epps et al. [13] 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 [14] 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

Figure 1. Trackballs come in many different sizes, making them appropriate as computer access and mobile technologies. From left to right: the Infogrip *BIGTrack* (http://www.infogrip.com), Kensington *Expert Mouse* (http://www.kensington.com), Appoint *Thumbelina* and Infogrip *Mini Trackball*. Relative image sizes are approximately maintained.

be less accurate than the mouse. Because of these findings, it was made sure that *Trackball EdgeWrite* does not require precise pointing or smooth freeform gestures like those in natural handwriting or most unistroke text entry methods (e.g. Graffiti, Jot).

An exercise similar to 'tracing' is 'steering'. Steering is the act of moving a mouse cursor or other pointing instrument along a path of a given width, like driving a car down a road. Accot and Zhai [15] studied five input devices, including trackballs, in various steering tasks. They found 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 noted that the performance of trackballs suffers when the device must be 'clutched' for travel over long distances. Accordingly, *Trackball EdgeWrite* avoids the need for clutching. That is, a user does not have to rotate the trackball farther then they are able without lifting their thumb from the ball.

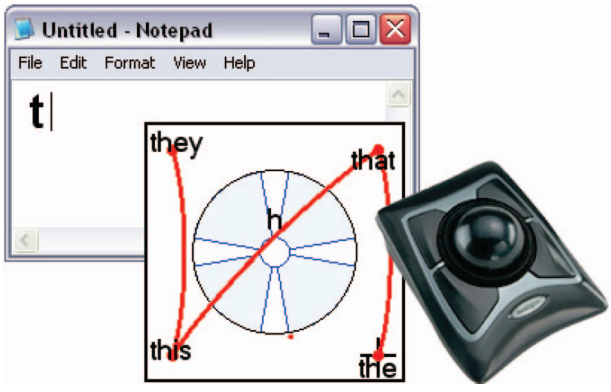

Figure 2. *Trackball EdgeWrite* with Microsoft Notepad. The Notepad window retains the input focus even while mouse movements are interpreted by *Trackball EdgeWrite*. Word completions beginning with 't' are offered at the corners of the EdgeWrite square.

Further comparisons of pointing devices by MacKenzie et al. [10] show that trackballs are slower than mice and styli in pointing and dragging and less accurate for dragging. Dragging is particularly difficult with a trackball because of the confluence of the thumb and finger muscles. In a later study, MacKenzie et al. [11] added that trackballs often move accidentally when clicking inside targets. Such slips can be particularly troubling for users with motor impairments [16,17]. Accordingly, *Trackball EdgeWrite* does not rely on buttons or require dragging or clicking. It can, in fact, be used button-free.

Hinckley and Sinclair [18] developed a touch-sensitive trackball called the *TouchTrackball*. When the *TouchTrackball* was touched, a ToolGlass window appeared in what the authors called an 'on-demand interface'. When the trackball was released, the ToolGlass faded away. Although adding touch-sensitivity to the design for *Trackball EdgeWrite* may have made certain features easier (e.g. knowing when to segment between letters), one wanted *Trackball EdgeWrite* to be usable with off-the-shelf trackballs that required no augmentation, thereby lowering barriers to access.

Trackball text entry

With the exception of *Trackball EdgeWrite*, there have been no text entry methods developed *explicitly* for trackballs. As noted above, most trackball text entry has been with on-screen keyboards. Example on-screen keyboards include *WiViK* [19] and the *Visual Keyboard* [20]. In Figure 3, WiViK is shown along with Microsoft Notepad.

While not explicitly designed for trackballs, two gestural text entry methods besides EdgeWrite can be used with them. The first of these is *Dasher*, a method that can be used with any cursor control device [21]. With Dasher (Figure 4), one moves

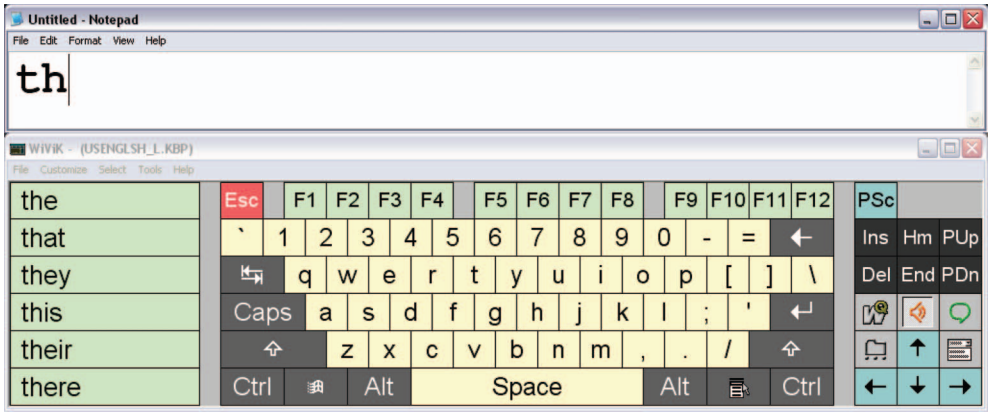

Figure 3. The WiViK on-screen keyboard with word prediction and completion activated. Preferences allow letters and words to be triggered by either clicking or dwelling.

through dynamically expanding letter regions, the sizes of which correspond to a letter's likelihood of entry. Although Dasher can achieve rapid entry rates (~30 wpm), a common sentiment, particularly among novices, is that it can be overwhelming because it is in constant visual flux. This also makes it difficult to use Dasher purely by feel, since one must constantly attend to its ever-changing display. Besides trackballs, Dasher can be used on a PDA with a stylus or with an eye-tracker for hands-free input [22].

A second trackball-specific text entry method is the Minimal Device Independent Text Input Method, or *MDITIM* [23]. MDITIM defines all letters in terms of *north*, *east*, *south* and *west* primitives (Figure 5). Like EdgeWrite, MDITIM can be used with trackballs and numerous other devices, including touchpads, mice, joysticks and keyboards. A potential downside of MDITIM is that letter shapes generally do not mimic their Roman counterparts in either look or feel. In contrast, the EdgeWrite alphabet (see Figure 7) was designed to maintain mnemonic correspondence with Roman letters to improve learnability.

In a study of MDITIM on different devices [23], subjects entered text with a touchpad for nine 30-minute sessions followed by a single session in which they used a touchpad, trackball, mouse, joystick and keyboard. Mean speeds (wpm) and error rates were ~7.5 (6.2%), 6.5 (7.3%), 6.3 (4.8%), 5.6 (3.0%) and 4.9 (3.2%), respectively. These speeds are quite a bit slower than EdgeWrite on the same devices [24]. Like Dasher, MDITIM has also been adapted for use with an eye-tracker, although no performance results were reported [25].

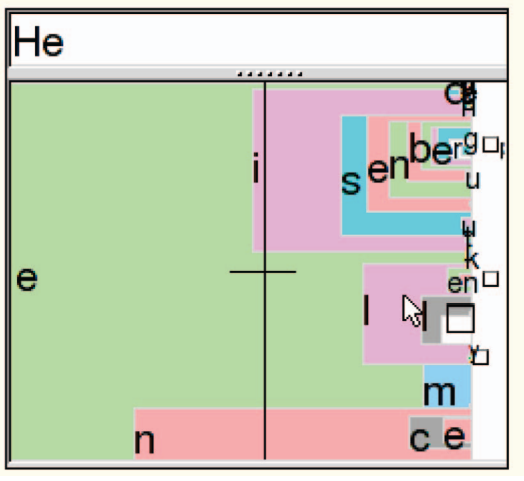

Figure 4. In Dasher, probabilistic letter regions expand rapidly from right-to-left toward the mouse cursor. Here the user is writing 'Hello'. Used with permission.

Word-level text entry

Researchers have noted that character-by-character text entry is inherently slow [26]. As a result, recent attention has shifted to *word-level* techniques. In word-level text entry, single gestures (or other operations) produce entire words at once. Cirrin [27] and Quikwriting [28] are two such techniques. In both designs, a person moves a stylus through fixed letter regions arranged around the periphery of a circle or square. These techniques are word-level in the sense that whole words are made in single (rather long) strokes. However, each character within the word must still be acquired by the stylus.

An innovative approach to word-level stroking is *SHARK* [26], which presents a stylus keyboard over which strokes can be made. The shapes of these strokes are determined by the arrangement of letters on the keyboard. Users can gradually 'ramp up' from tapping words to stroking them, enabling higher speeds. This emphasis on *gradual* transition from character-level to word-level entry has been preserved in *Trackball EdgeWrite*'s design for stroke-based word completion, since users can still stroke individual characters as they always have.

Word prediction and completion

A common approach to enhancing text input rates is to use word prediction and completion to populate a list with candidate words. Users select from the list to enter entire words or suffixes. Although the number of user actions is reduced, numerous studies show that additional perceptual and cognitive processes often make such systems *slower* instead of faster [29–31]. These findings highlight the challenge of

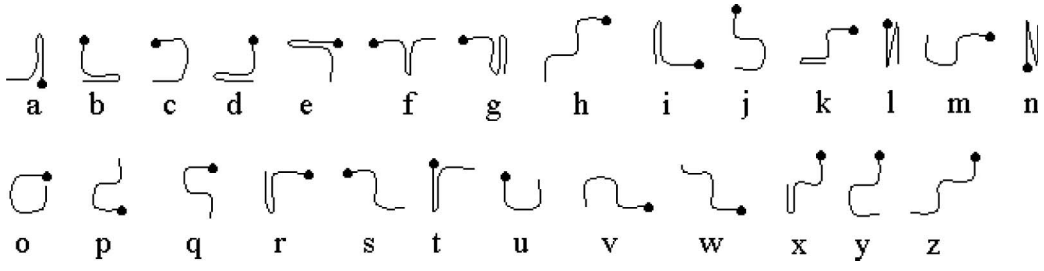

Figure 5. The MDITIM alphabet. Image adapted from MacKenzie and Soukoreff [9]. Used with permission.

designing effective word prediction and completions systems.

In the case of trackball text entry with on-screen keyboards, candidate words appear as additional mouse targets, which further exacerbate mouse travel and the need for accurate target pointing. Although Anson et al. [8] reported that word prediction and completion improved entry rates with on-screen keyboards, subjects reported high frustration because they disliked looking from their document to the word list and 'felt that searching through the word list was tedious and distracting'. As will be shown, stroke-based word completion in *Trackball EdgeWrite* overcomes the drawbacks of visually-intensive word selection by providing a 'feel-based' gestural alternative that performs just as well or better.

Now one turns to the design of *Trackball EdgeWrite*, including a detailed description of how the writing process works. This discussion is followed with a theoretical model of writing speed, after which empirical results are given.

The design of *Trackball EdgeWrite*

Background

EdgeWrite was originally designed to improve the accessibility of text entry on handheld devices like Palm PDAs [32]. Using on a Palm PDA, a user could move his or her stylus in Roman letter-like patterns along the edges and into the corners of a square bound by plastic edges (Figure 6). Stroke recognition was accomplished by simply looking at the *order* in which the corners of the EdgeWrite square were hit, which provided tolerance to wiggle since vagaries in the stroke path were not detrimental. The physical edges and corners provided tremulous users with additional physical stability, resulting in dramatic accuracy improvements over leading commercial products like Palm OS Graffiti, which is the built-in writing system for Palm PDAs. For example, one very tremulous subject with Parkinson's disease could enter text with only 30.6% accuracy in Graffiti, but with 94.4% accuracy in *Stylus EdgeWrite*. Similarly, the subject with a spinal cord injury described later in this article was 54.8% accurate with Graffiti 2 but 99.0% accurate with *Stylus EdgeWrite*.

It was clear that *Stylus EdgeWrite* was useful on PDAs, but a trackball is certainly very different from a stylus. The challenge was to take the alphabet used for *Stylus EdgeWrite* (Figure 7) and tailor it for writing with a trackball. An obvious hurdle, among others, was that the stylus version used 'lift' to segment between letters, but a trackball cannot be lifted in the same way. Thus, a different solution was required to solve the so-called 'segmentation problem' for determining where one letter-stroke ends and the next begins.

Figure 6. *Stylus EdgeWrite* on a Palm PDA device. The stylus moves along the edges and into the corners of the square hole, which provides stability for tremulous users.

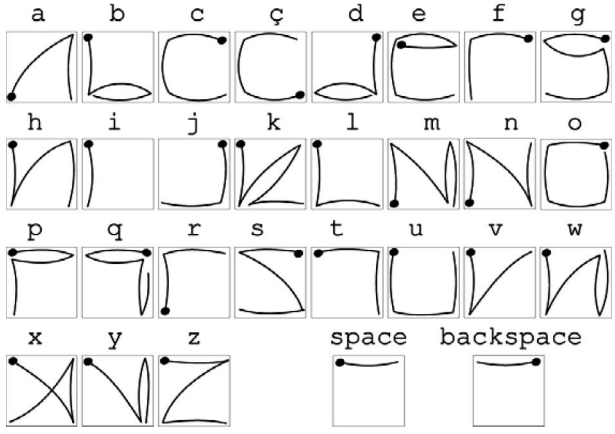

Figure 7. A chart showing the EdgeWrite letter strokes. Each letter is a single 'unistroke' from start to finish. Strokes begin at the heavy dot. The bowing of line segments is for illustrative purposes only; all actual motion is ideally in straight lines along edges and into corners.

Initial design: Using impenetrable virtual edges

The first design for *Trackball EdgeWrite* essentially replicated the physical design for *Stylus EdgeWrite* in a virtual space.[2] A window with *impenetrable virtual edges* was used to constrain the movement of the mouse cursor during writing. The cursor could then be moved from corner to corner within this bounded virtual square (Figure 8(*a*)), much like the stylus itself moved inside a plastic square for *Stylus EdgeWrite*.

In an effort to make the cursor's movement more accurate, a movement correction scheme was used to move the cursor directly toward the user's current 'intended' corner. Figure 8(*b*) and equation (1) show how this correction was applied. A visible virtual

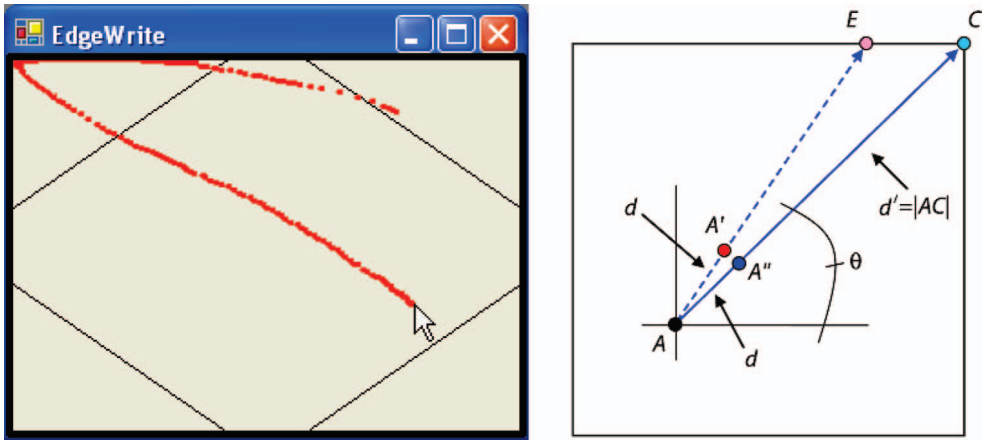

Figure 8. (*a*) The initial design for *Trackball EdgeWrite* used impenetrable virtual edges, a visible virtual cursor and a hidden underlying system cursor. (*b*) The attempt to make cursor movement more accurate by inferring the user's 'intended corner' and always moving directly toward it.

cursor, the one the user sees, and the underlying hidden system cursor are both positioned at point A. As the user moves the trackball, the invisible system cursor is moved a distance d to point A'. A projection along the path from A to A' intersects the square's edge at point E, to which the nearest corner is point C. The distance d is then applied to the visible virtual cursor at A to move it to point A'', which is on a straight-line to C. Point A'' is computed using equation (1), in which capital letters are points with (x,y) components and d and d' are distance scalars (d' is the distance from A to C). Lastly, the underlying system cursor at A' is realigned with the virtual cursor now at A''. This is done continuously for all movements within the virtual square, allowing the user to change directions instantly but always be on a straight line to their 'intended' corner.

$$A'' = \left(A_x + \frac{d}{d'}(C_x - A_x),\quad A_y + \frac{d}{d'}(C_y - A_y)\right) \quad (1)$$

Admittedly, the use of impenetrable virtual edges was an obvious starting point for the design of *Trackball EdgeWrite*, since *Stylus EdgeWrite* had used impenetrable physical edges. However, although the straight-line movement scheme helped, in the end this design for *Trackball EdgeWrite* did not 'feel right'. This was because there was only a loose correspondence between a user's motion on the physical trackball device and the location of the mouse cursor within the virtual EdgeWrite square. In other words, these did not always tightly correspond. For example, if the user rotated the physical trackball towards the top-left, there was no guarantee that the mouse cursor would actually *be* in the top-left corner – it would depend on where the mouse cursor started. Thus, another approach was necessary, one that more tightly coupled users' movements on the physical trackball with their mouse cursor's location in the EdgeWrite square. The solution that worked was *goal crossing*, described in the next section.

Making characters with goal crossing

Target pointing, or just 'pointing' for short, has been a fundamental aspect of graphical user interfaces for over 25 years. Pointing has been well-studied, in large part because it can be rigorously modelled by Fitts' law, at least for able-bodied individuals [33]. Fitts' law quantifies the time required to access a target as a function of the target size W and its distance away A [33]. Not surprisingly, it takes longer to successfully click on small and/or distant targets. Figure 9 illustrates the Fitts' law parameters. Equation (2) is a commonly used formulation of Fitts' law [34].

$$T = a + b \cdot \log_2\left(\frac{A}{W} + 1\right) \quad (2)$$

In equation (2), a and b are empirical coefficients determined by linear regression. A powerful aspect of Fitts' law is that the units (e.g. pixels, mm, etc.) of A and W cancel, allowing researchers to compare results from different experiments that used different values for A and W. The log term is often called the *index of difficulty* (*ID*).

One drawback of the initial design of *Trackball EdgeWrite* described in the previous section was that it *felt too much like pointing*. Although pointing is a successful interaction technique for numerous computer-based operations, it does not lend itself to fluid gesture-based writing. In fact, on-screen keyboards are tedious to use in large part because they are too dependant upon pointing.

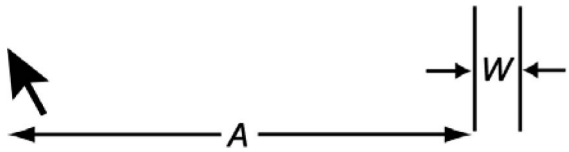

Figure 9. Fitts' law models the time required to access a target of size W at a distance A units away.

Figure 10. Crossing follows Fitts' law but with the W constraint orthogonal to the movement trajectory, rather than collinear.

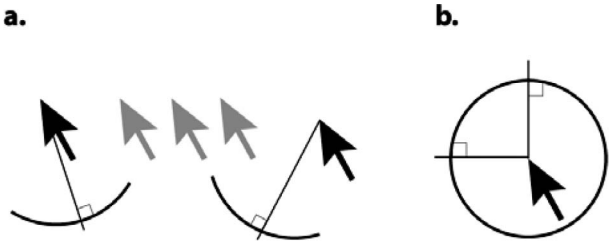

Figure 11. Accot and Zhai speculated that crossing goals could rotate to always remain orthogonal to the cursor, thereby offering maximum target width. An extreme form of this idea is a cursor in the centre of a circle, where the circumference itself is the goal.

An alternative to target pointing is *goal crossing* [35]. Unlike pointing, which requires a cursor to enter an area target and remain inside it long enough to select it, goal crossing requires only that a goal line be crossed, much like an American football player scoring a touchdown (Figure 10). One can think of the goal line as providing a similar benefit as an impenetrable edge. In both cases, the cursor is allowed to overshoot the target by an arbitrary amount and still successfully acquire the target.

Accot and Zhai [35] have shown that crossing follows the Fitts' law formulation shown in equation (2), but that the regression coefficients a and b differ than for the traditional Fitts' reciprocal pointing task [36]. In particular, for *ID*s less than 4-bits, continuous orthogonal goal crossing is faster than pointing. Currently, motion in *Trackball EdgeWrite* uses *ID*s ranging from $0.7-0.9$ bits, where crossing performs very well.

Accot and Zhai [36] compared different types of pointing and crossing using a stylus. They found that the fastest arrangement for short-range reciprocal trials 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. They speculated that goals could rotate to always remain orthogonal to the cursor and thus offer the maximum target width (Figure 11(a)). An extreme form of this idea is a cursor placed inside a circle, where the circumference is the goal (Figure 11(b)). This insight is one inspiration for the design of *Trackball EdgeWrite*.

Using *Trackball EdgeWrite*, one writes by making short 'pulses' on the trackball towards intended corners. When these pulses cause the mouse cursor to cross the circumference of a circle, the resultant angle indicates the next corner. Thus, the cursor does not actually *travel* among corners to acquire them as targets as in the original scheme, but crosses a radius and 'snaps' to the next corner instantly. In essence, it is the *vector* along which the cursor moves that determines the next corner. Figure 12 depicts this process for writing the letter 'z'.

In Figure 12(a), 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 (Figure 12(b)). 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 (Figure 12(c)). Note that for the third stroke, the cursor moves towards the *outside* of the virtual EdgeWrite square, but at an angle that still indicates the bottom-right corner. The completed 'z' is thus drawn (Figure 12(d)).

Thus, a series of short crossing tasks forms a letter. However, clearly, much depends on how the circle's angular space is partitioned and how θ is interpreted. For instance, consider the move across the bottom of the 'z' in Figure 12(c). At what angle θ should the cursor instead move diagonally up to the top-right?

Figure 13 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 virtual EdgeWrite square.

Three features of the design in Figure 13 are important. First, any movement *left*, diagonal *down-and-left* or *down* from the centre of the circle keeps the cursor fixed in-place (i.e. in 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 at that corner to begin with. This will always hold true for $(180-\theta_d)^\circ$ of the circle, where θ_d is the angular amount allotted to each diagonal. This 'pinning' of the cursor to the circle's centre ensures that one always starts from a corner's centre when moving to a different corner.

Secondly, the value in having big cardinal angles (θ_c) is that users do not always move perfectly to the

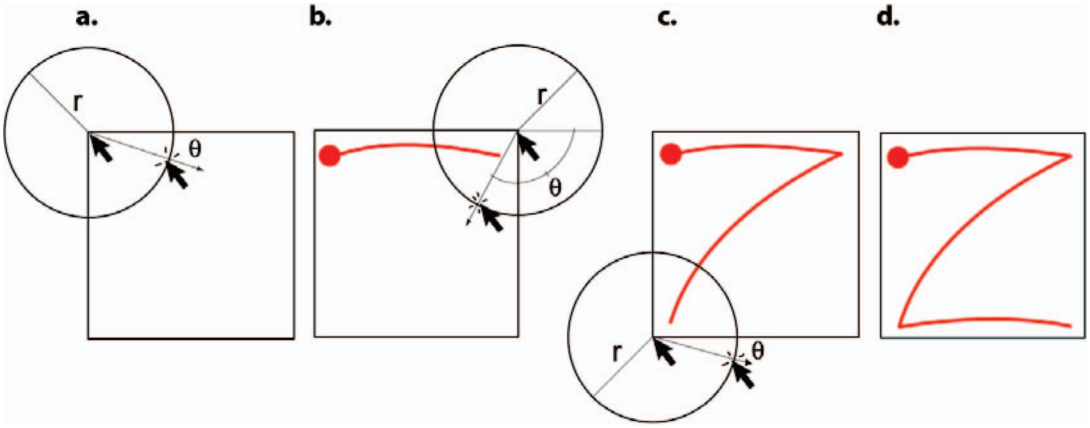

Figure 12. The repeated goal crossings involved in writing the letter 'z'. Each successive corner is determined by the current corner and the angle at which the crossing occurs.

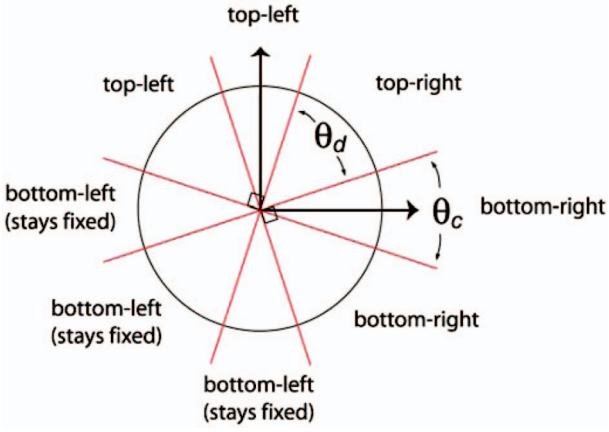

Figure 13. Next-corner outcomes for different angles of departure from the bottom-left corner of the EdgeWrite square. θ_d is the diagonal angle and θ_c is the cardinal angle.

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 $\theta_c + \theta_d = 90^\circ$.

Thirdly, 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. This is because one always has the angles to the *outside* of the EdgeWrite square, as in Figure 12(*c*). So, although θ_d and θ_c are tradeoffs, a total of 90° remains for moving along an edge. In practice, however, having a larger θ_c provides a 'cardinal error band', giving more room to move inside the EdgeWrite square.

It is important to emphasize that, despite these underlying mechanics, users neither see a mouse cursor nor 'aim' for particular angles when writing with *Trackball EdgeWrite*. Instead, users simply 'pulse' the ball towards intended corners with small movements of the trackball, creating the feeling of fluid gestural writing.

Determining the first corner

The example in Figure 12 assumes that one starts in the top-left corner. However, of course, not all EdgeWrite letters start there. This begs the question of how one *begins* a letter, since, unlike a stylus landing on a PDA, the underlying mouse cursor is persistent and cannot simply 'appear' in the starting corner.

Two different schemes were designed and implemented for determining the first corner in *Trackball EdgeWrite*. The first scheme is to have users enter the initial corner as if they were starting from the centre of the EdgeWrite square (Figure 14(*a*)). 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 the starting corner.

In the second scheme, users assume they *start* in the appropriate corner and the software disambiguates that corner as the gesture unfolds (Figure 14(*b*)). For example, in making a 'z', one would first move to the right. At this point, one may have intended to move along either the top or bottom edge, so both possibilities are entertained. The next move is diagonally *down-and-left*, at which point the ambiguity is resolved. Although gestures that occur along a single edge (e.g. 'i', space, backspace) never resolve into unambiguous strokes, such gestures can be defined identically for both sides of the square, rendering the ambiguity irrelevant.

Although the second scheme requires one less pulse per letter than the first scheme, it proves slower and more difficult in practice because the initial stroke has *eight* possible outcomes (Figure 15), not just four as in the first scheme. Even when the angular regions of the circle are adjusted to be proportional to the probability of beginning a letter in that region, the second scheme proves too difficult to perform reliably.

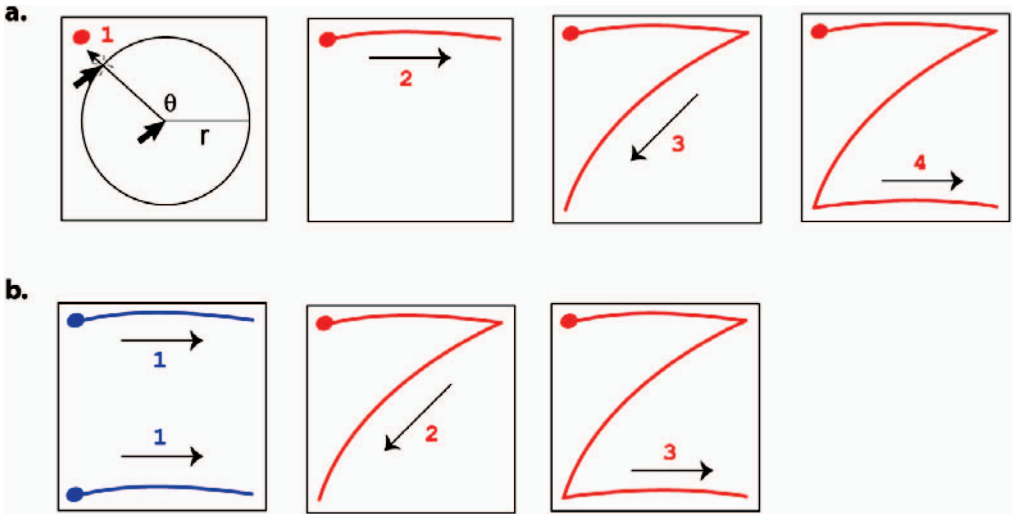

Figure 14. Two different schemes for determining the first corner.

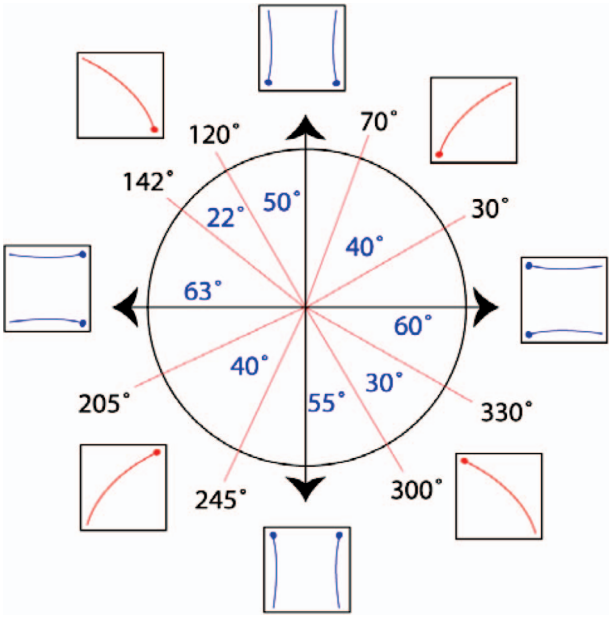

Figure 15. With the second first-corner scheme, users have eight angular regions from which to choose on the first pulse of the trackball. Region sizes are weighted by letter probabilities for letters that begin in them.

An interesting result is that a theoretical analysis of each scheme actually shows the first to be about 3 wpm *faster* than the second despite the additional stroke required for each letter. This is because of the high accuracy required for the second scheme's initial stroke. Thus, the first scheme was preferred. This was an unexpected theoretical finding, as it had initially been assumed the second scheme would be ~30% faster based on the reduced number of trackball movements required per letter. That both empirical and theoretical results proved otherwise shows the importance of basing designs on quantitative and theoretical measures instead of mere intuition.

Word-level stroking

Even the most efficient character-level unistroke systems are slow because they enter only one character at a time [37]. Unlike touch-typing with multiple fingers, unistroke methods do not support parallelism in the input task. This inherently limits the entry rate. One way to increase the input rate is to increase the efficiency of actions. For example, instead of one stroke producing one letter, one stroke can produce one *word*. A problem with defining a host of word-level strokes, however, is the myriad of strokes required and the challenge of learning them. Unlike letters, words are not easily represented by 'mnemonic' Roman letter-like stokes.

Along with *Stylus EdgeWrite* [38], *Trackball EdgeWrite* provides both character-level and word-level entry at the same time and in the same space. Entire words can be entered in a very few strokes by extending character-level strokes in minimal ways, but character-level strokes themselves remain unchanged. This enables users to gradually 'ramp up' from the more familiar character-level strokes to using word-level strokes. Importantly, the same stroke always produces the same word, enabling users to memorize the strokes at the motor level. Since natural language follows Zipf's law [39], it is conceivable that learning a small number of common words will increase users' overall text entry rates. For example, the word 'the' represents over 6% of the British National Corpus and the most common 100 words account for over 46% [26].

When a user strokes, candidate words are shown at the four corners of the EdgeWrite square (Figure 16). In order to provide word completions,

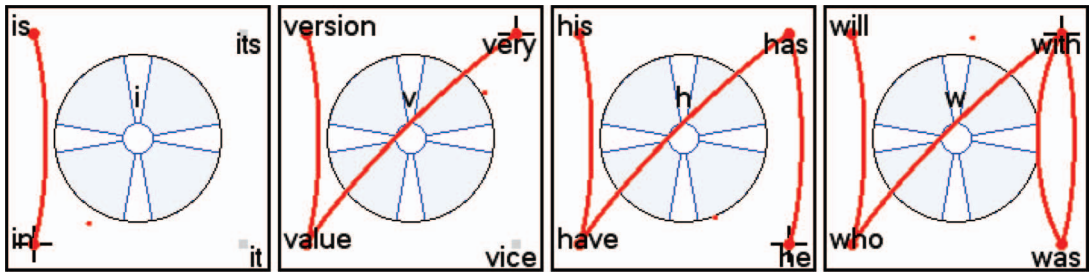

Figure 16. Candidate words shown while stroking a 'w' include words that begin with an 'i', 'v' and 'h' along the way, since these letters are all sub-sets of 'w'.

the current stroke is recognized after each corner is entered. This is called *continuous recognition feedback*, since this technique also displays the current stroke result in the centre of the EdgeWrite square. Thus, the user knows what his or her stroke will produce before the stroke is segmented. If the user slips, he or she can simply restart the stroke before segmenting using a feature called *non-recognition retry* [40].

After each letter is finished and a letter is produced, the user can continue stroking letters or, alternatively, make a short gesture to select a word. This gesture used to select word completions is a short motion from the centre of the EdgeWrite square to the corner containing the desired word. After the completion of this gesture, the word at that corner is entered.

When the text cursor is positioned after a word that has not been completed, a word-level backspace from right-to-left across the bottom of the EdgeWrite square erases the entire previous word. However, if the previous word was composed using word completion, then a word-level backspace along the bottom of the EdgeWrite square will erase *only* the completed suffix, restoring the word completions as they appeared before selection. Importantly, the restored completions appear in the *same* corners as before, allowing the user to quickly select a different completion if desired. Completed words always remember the character position at which they were completed (if any), allowing future word-level backspace strokes to remove completed suffixes. Thus, completions are quickly undoable and re-doable.

An important aspect of this design is that the same completion is always shown in the same corner for the same prefix. This is because completions are based only on English word frequencies, not on context. This consistency is important for enabling users to rely on the positions of words. For example, after stroking a 't', the word 'the' is *always* shown in the lower-right corner. Thus, users can come to rely on the position of 'the' and stroke it 'by feel' rather than 'by sight'. The consistency of word positions also reduces cognitive load as motor performance comes to dominate.

In addition to showing frequency-based word completions, *Trackball EdgeWrite* also shows context-dependent word predictions after a word ends. Word predictions are, by definition, based on surrounding context and thus cannot be stroked by feel.

Trackball EdgeWrite's design for word-level stroking avoids high perceptual search times by showing only four words at a time, generally less than most word completion systems [30]. However, how useful are only four words? To answer this question, a computer program was written to calculate the amount of 'language coverage' obtained for 1–5 letter prefixes showing only four frequency-based word completions per letter (Figure 17). Kucera-Francis frequencies were used for the 17 805 most common English words [41]. According to the graph, users have a 49.0% chance of seeing their intended word after just one letter. After two letters, this climbs to 70.8%. After three, it's 89.3%. This is the Zipf's law effect at work [39].

As explained elsewhere [38], one can achieve a slightly higher language coverage by not re-showing the same word completions once they have been shown for a given word being entered. For example, when 't' is written, 'the' is one possible completion. If 'the' is not selected and an 'h' is written next, should 'the' be re-shown? Or, since the user did not select 'the', should a different word be shown in its place, gaining more coverage? In *Trackball EdgeWrite*, previously shown words are reshown because users sometimes miss the initial appearance of the word they want, especially when first learning to use the system.

Application design

Thus far, much attention has been given to the underlying writing mechanics in *Trackball EdgeWrite*. However, *Trackball EdgeWrite* is a full-fledged desktop application and, as such, has other important aspects of its design.

One important part of *Trackball EdgeWrite* is how users segment between letters. As noted above, segmenting between letters in *Stylus EdgeWrite* was

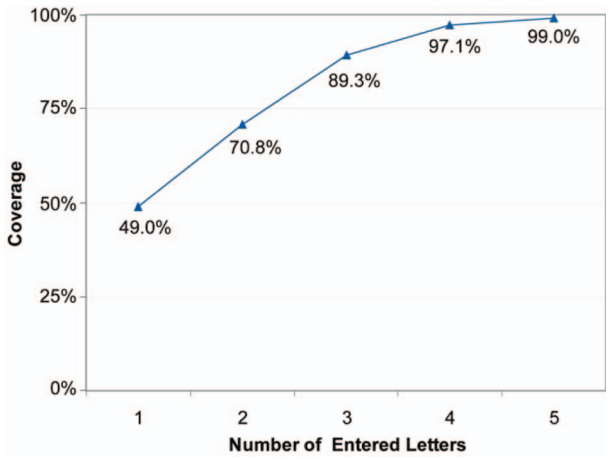

Figure 17. Coverage of the 17 805 most common English words [41] based on 1–5 letter prefixes and four frequency-based completions shown per entered letter.

accomplished by simply lifting the stylus. This is not possible in *Trackball EdgeWrite*. Instead, users segment when the underlying mouse cursor stops moving for a very short period of time. Timeout values can be set in the preferences dialogue (Figure 18) and range from 100 ms for experts to 750 ms for novices. This simple scheme works reliably since the timer is restarted after every mouse movement. A beginner can still 'stop and think' while making a gesture by slowly rolling the trackball toward the corner they are already in. (Recall that this will not move the cursor to a new corner, but it will prevent the stroke from segmenting.) Subjectively for the user, segmentation involves a slight pause between letters. In pilot studies, users had no trouble segmenting letters.

Another important aspect of *Trackball EdgeWrite* is how users switch between mousing and entering text, since both are accomplished using the trackball. *Trackball EdgeWrite* is designed to run invisibly in the background until it is needed for text entry. When the trackball is being used for mousing, the mouse cursor is said to be 'released'. When the trackball is being used for writing, the cursor 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 designated corner of the desktop (a 'hot corner'). Hot buttons, hot keys and hot corners are all configurable in the application's preferences dialogue (Figure 18).

When captured, the mouse cursor vanishes and the *Trackball EdgeWrite* window appears (Figure 19). To release the cursor, the user can click any trackball button or perform a dedicated *release gesture*. Then the EdgeWrite window disappears and the mouse cursor is restored to its position before being captured. Thus, *Trackball EdgeWrite* never requires that it be navigated to; instead, it comes to the user when summoned and departs quickly when dismissed.

Other preferences exist for setting performance parameters like the radius of the crossing circle, the amount of diagonal degrees, the segmentation timeout and the underlying mouse sensitivity. Additional options exist for controlling window transparency, playing sounds and determining where the EdgeWrite window appears.

Implementation

Trackball EdgeWrite is implemented in Visual C# using DirectInput 9.0c to receive mouse events in the background, which is necessary so that focus can remain on a target application (e.g. Microsoft Notepad) even while *Trackball EdgeWrite* receives mouse input. A fair amount of code in *Trackball EdgeWrite* is devoted to keeping the input focus on the target window and *off Trackball EdgeWrite* itself. Cases that must be handled include when the user left-clicks while captured or left-drags to reposition the *Trackball EdgeWrite* window. After these actions, focus is returned to the target window.

Recognized characters and words are sent through the low-level input stream as if they were typed on the computer's physical keyboard. *Trackball EdgeWrite* works with any pointing device, but is best suited for devices without absolute position (e.g. trackballs and isometric joysticks). With a miniature trackball or an isometric joystick, EdgeWrite provides full text entry in very little physical space.

Trackball EdgeWrite's word-level entry system has four components: (1) a vocabulary list of words and frequencies, (2) an optional user-defined vocabulary list, (3) a trigram list with trigram frequencies and (4) an adaptive bigram cache that stores users' words at runtime. The first and second items provide 'fixed' English frequency-based word completions as words are being made. The third and fourth items provide context-dependent word predictions after a word has been completed (i.e. after a space has been entered).

The vocabulary list is stored in an alphabetically sorted array enabling binary search for fast lookups. Each array slot contains a word string and the word's frequency count. This is all the data necessary for the fixed English frequency-based word completions shown while users stroke letters (see Figure 16).

Also in each slot of the vocabulary array is a hash table whose *keys* are word indices and whose *values* are a list of word indices. These indices correspond to slots in the vocabulary array. A slot's word string represents the first word of a trigram, its hash table keys represent second words and its hash table list values represent third words. These data structures

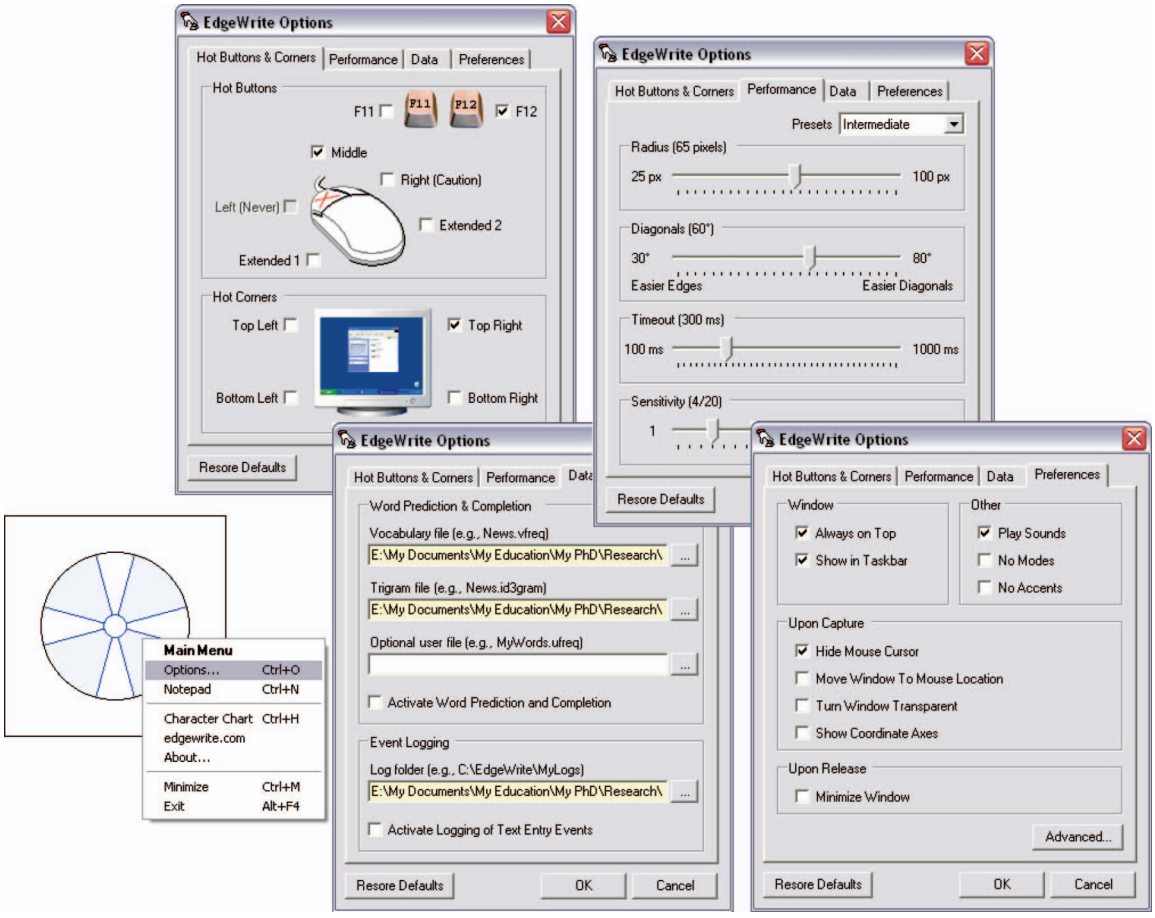

Figure 18. *Trackball EdgeWrite* supports multiple options for capturing the trackball with hot buttons, hot keys and hot corners (top left). Other options are also available for controlling the parameters of the writing task (e.g. the segmentation timeout).

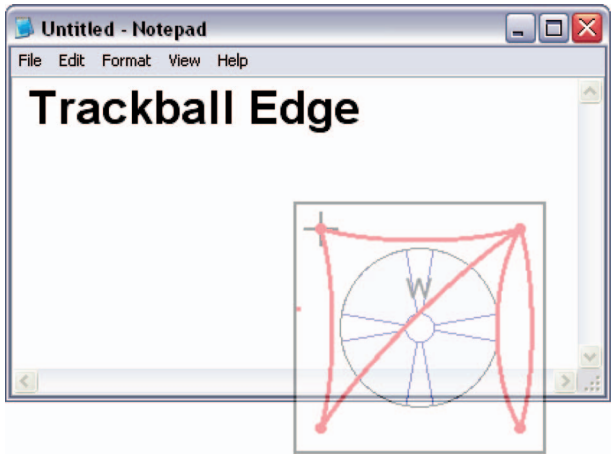

Figure 19. A semi-transparent *Trackball EdgeWrite* is being used with Microsoft Notepad. Notepad retains the input focus even though EdgeWrite receives mouse events.

allow fast lookups for both fixed completions and context-dependent word predictions.

When a letter is entered, words that begin with the current prefix are gathered from the vocabulary list. If a user-defined vocabulary list is loaded, its words with matching prefixes are also gathered. These words are then sorted in a separate list according to their frequencies. The top four words are then offered as completions (Figure 16). Since frequencies are pre-computed based on a large corpus of English words, these four completions will always be the same for a given prefix, allowing word positions to become familiar to users.

When four English frequency-based words are retrieved from the language model, they are assigned to corners such that the highest priority word is given the corner in which the current stroke resides. The two adjacent corners receive the next two words and the lowest priority word is placed at the diagonal away from the stroke's current corner. Once a word has been shown, it is stored in a hash table along with its corner and a 'half-life'. If a word is shown again, it will be shown in the same corner as it was before. If the word goes unused for some time, it will 'decay' and be eligible for reassignment. If a collision occurs with two words vying for the same corner, the highest priority word wins.

After a space is entered (i.e. between words), context-dependent predictions are offered. The most

recent two words are used to look up possible third-word predictions. The first word is found in the vocabulary array using binary search. The second word's index, which was found when the word was entered, is used as a hash key in first word's hash table for fast lookups. The value returned, if any, is a linked list of possible third words. The top four from this list are shown as predictions.

Predictions may also come from an adaptive bigram cache holding users' recently entered words. This cache holds word-pairs so that when a user enters a previously used word, words that followed it can be offered as predictions. The cache is a list maintained in priority order such that when a new bigram is entered or an old bigram is reused, it is placed at the top. Unlike the trigrams, the adaptive bigram cache accommodates out-of-vocabulary words, enabling the prediction of last names from first names, common phrases and so on.

The English vocabulary list and trigrams were built by parsing 850 MB of news articles from the *Wall Street Journal*, Ziff Davis, *Los Angeles Times* and Associated Press. This parsing was carried out with the CMU-Cambridge Statistical Language Modelling toolkit [42]. Custom parsers then pared down the toolkit's results, keeping 20 000 of the most common words and only trigrams that occurred at least 20 times. After certain abbreviations were removed, the result was a 258 KB vocabulary list of 19 122 words with frequency counts totalling 132 701 943. The maximum frequency count was for the word 'the' at 7 686 122, or 5.79%. The trigram list is 10.6 MB and contains 517 988 trigrams with frequency counts totalling 40 230 622. The maximum frequency count is for the trigram 'the United States', at 46 947 or 0.12%. Although news articles were parsed, this procedure could easily be run over other corpora (e.g. email, instant messaging, academic prose, other Western languages, etc.).

Evaluation

This section presents two separate evaluations of *Trackball EdgeWrite*. The first evaluation quantifies the theoretical speed of the method based on an underlying model of goal crossing. The second evaluation is empirical and involves an extended field study of a user with a spinal cord injury. This user has used a trackball for 15 years and has, until now, relied on on-screen keyboards as a key part of his text entry solution. Now he uses *Trackball EdgeWrite* instead, preferring it to any of the on-screen keyboards he has used. Results for his performance are presented for both character-level and word-level *Trackball EdgeWrite*.

Theoretical model

The time it takes to make a single letter can be predicted using Accot and Zhai's [35] discovery that crossing follows the Fitts' formulation in equation (2).

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

$$W_d = \frac{\theta_d}{360} \cdot 2\pi r \tag{3}$$

The index of difficulty for diagonal movement ID_d is therefore:

$$ID_d = \log_2\left(\frac{r}{\frac{\theta_d}{360} \cdot 2\pi r} + 1\right) \tag{4}$$

Thus, the time for diagonal movement T_d is given by:

$$T_d = a + b \cdot \log_2\left(\frac{180}{\theta_d \cdot \pi} + 1\right) \tag{5}$$

Recall that a and b are coefficients determined by linear regression according to Fitts' law (equation 2).

For movement in the cardinal directions, recall that there is a constant target size of 90°. The target width W_c is therefore:

$$W_c = \frac{90}{360} \cdot 2\pi r \tag{6}$$

So, the index of difficulty for cardinal movement ID_c is:

$$ID_c = \log_2\left(\frac{r}{0.5\pi r} + 1\right) \tag{7}$$

The movement time along cardinal edges T_c is thus given by:

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

For the pulse from the centre to the first corner, there are four target angles each of 90°. Thus, the movement time T_f for the pulse to the first corner is the same as T_c:

$$T_f = a + b \cdot 0.710719 \tag{9}$$

Using equations (5), (8) and (9), the theoretical movement time for each EdgeWrite letter and the space character can be computed. For example, using a typical 65° for diagonal angles θ_d and 150 ms for segmentation timeout τ, the time (in ms) to enter the letter 'z' is given by:

$$\begin{aligned} T_{\text{'z'}} &= T_f + T_c + T_d + T_c + \tau \\ &= 3 \cdot T_c + T_d + \tau \\ &= 4a + 3.04402b + 150 \end{aligned} \tag{10}$$

Obviously, the result for $T_{\text{'z'}}$ (or any other letter) is dependent upon the choice of a and b – the Fitts' regression coefficients. To the authors' knowledge, no studies have elicited these coefficients for continuous reciprocal goal crossing with a trackball. However, Accot and Zhai [36] have done so for a stylus One can use these studies to estimate a and b for continuous reciprocal goal crossing with a trackball. This study uses $a = -363.0$ and $b = 642.1$.

To compute the time (in ms) of the 'average character' T_{avg}, each character's time T_i is weighted by its linguistic frequency F_i.

$$T_{\text{avg}} = \sum_{i \in C} T_i \cdot F_i \tag{11}$$

In equation (11), i is a character in character set C. For simplicity, the primary letters shown in Figure 7 are used for C, omitting 'ç' and backspace but including space. Next, equation (12) is used to calculate the theoretical speed (wpm) for the character-level method (the numerator is ms min^{-1}):

$$\text{wpm} = \frac{60,000}{5 \cdot T_{\text{avg}}} \tag{12}$$

Using a typical $\theta_d = 65^\circ$ and $\tau = 150$ ms, equation (12) yields 23.1 wpm as the theoretical character-level speed for *Trackball EdgeWrite*. Note that this rate represents 'perfect' entry – 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, this estimate can only be considered a ballpark measure. However, it is useful for comparing different stroking schemes, such as the two first-corner schemes discussed earlier.

This theoretical model can be extended to incorporate *Trackball EdgeWrite*'s frequency-based word completions. To do this, a computer program was needed to calculate the speed of each word in *Trackball EdgeWrite*'s list of 19 122 words, assuming that each completion is selected by the user when it first appears. This word list is large enough to accommodate most words used in everyday English.

The speed S_{cps} for this corpus can be calculated using equation (13):

$$S_{cps} = \sum_{w \in K} \left(\frac{|w| + 1}{T_w} \times F_w \right) \times 1000 \tag{13}$$

Here, S_{cps} is the weighted speed of text entry in characters per second (cps), w is a word in corpus K with length $|w|$, T_w is the time to write word w (in ms) and F_w is the frequency of word w such that $\Sigma F_w = 1.00$. The '+1' in the numerator is for the space that is added after a completion is selected, and the '$\times 1000$' converts from characters per ms to cps.

To calculate T_w (in ms) for each word in the corpus, the time T_ℓ to perform each letter was calculated $\ell \in w_p$, where w_p is the minimum prefix that will show w as a completion ($1 \le |w_p| \le |w|$). To this one adds T_{select}, the time to select the completion itself, which is equivalent to T_f (equation 9). As in equation (10), $\tau = 150$ ms must be included in the sums for T_ℓ and T_{select} to account for the segmentation time after a letter or word selection is made. Note that the computation of T_ℓ is akin to equation (10) for each letter (a–z) and space. Thus, the time T_w to write word w is:

$$T_w = \left(\sum_{\ell \in w_p} T_\ell \right) + T_{\text{select}} \tag{14}$$

For words which themselves are prefixes of at least four other more common words,[3] there is no w_p that will show w as a completion. For such words, w must be entered character-by-character along with a trailing space:

$$T_w = \left(\sum_{\ell \in w} T_\ell \right) + T_{\text{space}} \tag{15}$$

To convert S_{cps} in equation (13) from cps to wpm, one uses the standard definition of five characters per word:

$$S_{wpm} = S_{cps} \times \frac{60\,\text{sec}}{1\text{min}} \times \frac{1\ \text{word}}{5\ \text{chars}} \tag{16}$$

Using equations (13–16), this word-level model yields an upper-bound entry rate of 52.5 wpm. This is 227% faster than the 23.1 wpm obtainable with only character-level strokes. Like before, this result is unachievable by a real human user. It represents 'perfect entry', lacking considerations for hesitation, cognitive processes, visual search time, slips or mistakes. Still, it is useful as an upper-bound for theoretical comparisons with prior models and alternative designs.

For a more realistic estimate, the model can be enriched with a term for visual search time based on the Hick-Hyman law [43,44]. Although this law is not specific to word-reading, for these purposes it will provide a sufficient estimate. A term for visual search time T_n is added after the entry of every letter ℓ and represents the time it takes for a user to spot their word amidst n choices, where n is the number of completions offered for the current prefix ($0 \le n \le 4$). Drawing on prior rationale [45], the formula for T_n (in ms) is:

$$T_n = 0.2 \times \log_2(n) \times 1000 \tag{17}$$

Incorporating the Hick-Hyman law, equations (14) and (15) become:

$$T_w = \left(\sum_{\ell \in w_p} (T_\ell + T_n) \right) + T_{\text{select}} \qquad (18)$$

$$T_w = \left(\sum_{\ell \in w} (T_\ell + T_n) \right) + T_{\text{space}} \qquad (19)$$

Using equations (17–19), the result drops 36.2% from 52.5 wpm to 33.5 wpm – 45.0% faster than the character-level result of 23.1 wpm. As it will be seen, this is similar to the empirical performance gain of the trackball user. Note that even with the addition of visual search time T_n, this result still represents 'perfect entry'.

A limitation of this model is that it only accounts for word completion, not word prediction. However, modelling word prediction is more difficult because it depends on context, including the user's adaptive cache of recent words. Such a model is beyond the current scope of this theoretical analysis.

Field study

To evaluate *Trackball EdgeWrite*'s effectiveness over a period of time and to obtain feedback from a real user which could be used to improve the software, a multi-session study was run with one trackball user whom will be called 'Jim'. Although using multiple subjects in a single session has certain benefits, it was decided instead to 'go deep' with one user over multiple sessions to progressively inform the iterative design process.

Jim has a spinal cord injury that reduces the dexterity in his arms, hands and fingers such that he cannot satisfyingly use a conventional keyboard or mouse. For 15 years he has relied upon trackballs and on-screen keyboards for computer access. His favourite trackball is the *Stingray* from CoStar Corporation (Figure 20).

Jim is 46 years old and is still an avid trackball user. However, 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 suffer. 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 just 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 data and filling out web forms.

Figure 20. Jim's favourite trackball is the CoStar Stingray because he can press its wide left button with the palm of his left hand while his left thumb rolls the ball to perform dragging operations, which are traditionally difficult with trackballs.

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. His current on-screen keyboard is the Microsoft Accessibility Keyboard.

Jim has three complaints about using an on-screen keyboard. First, he feels that the visual attention required is enormous, as he constantly must look from the keyboard to his document and back. Secondly, 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 500 ms as the dwell time. Thirdly, 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 other findings [29,31,46]. Plus, word prediction adds more items for Jim to visually scan, adding to the tedium [8].

Character-level study. To ensure that the design iterations were beneficial, short 'checkpoint studies' were conducted when meeting with Jim. During these studies, Jim would enter 5–7 test phrases with *Trackball EdgeWrite* and with his preferred on-screen keyboard, the Microsoft Accessibility Keyboard (MAK). Thus, the studies were a single-subject 1-factor design, with a factor for *Method* (MAK, EdgeWrite). Word prediction was not available in either method at this stage. Both methods were controlled by Jim's personal trackball (Figure 20) and neither method required any button presses. Jim's speed and accuracy results are shown over numerous weeks in Figure 21.

Note that the checkpoint studies were not evenly spaced in time, particularly in the jump from week 4 to week 10 and week 22 to week 32. During the first gap, 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

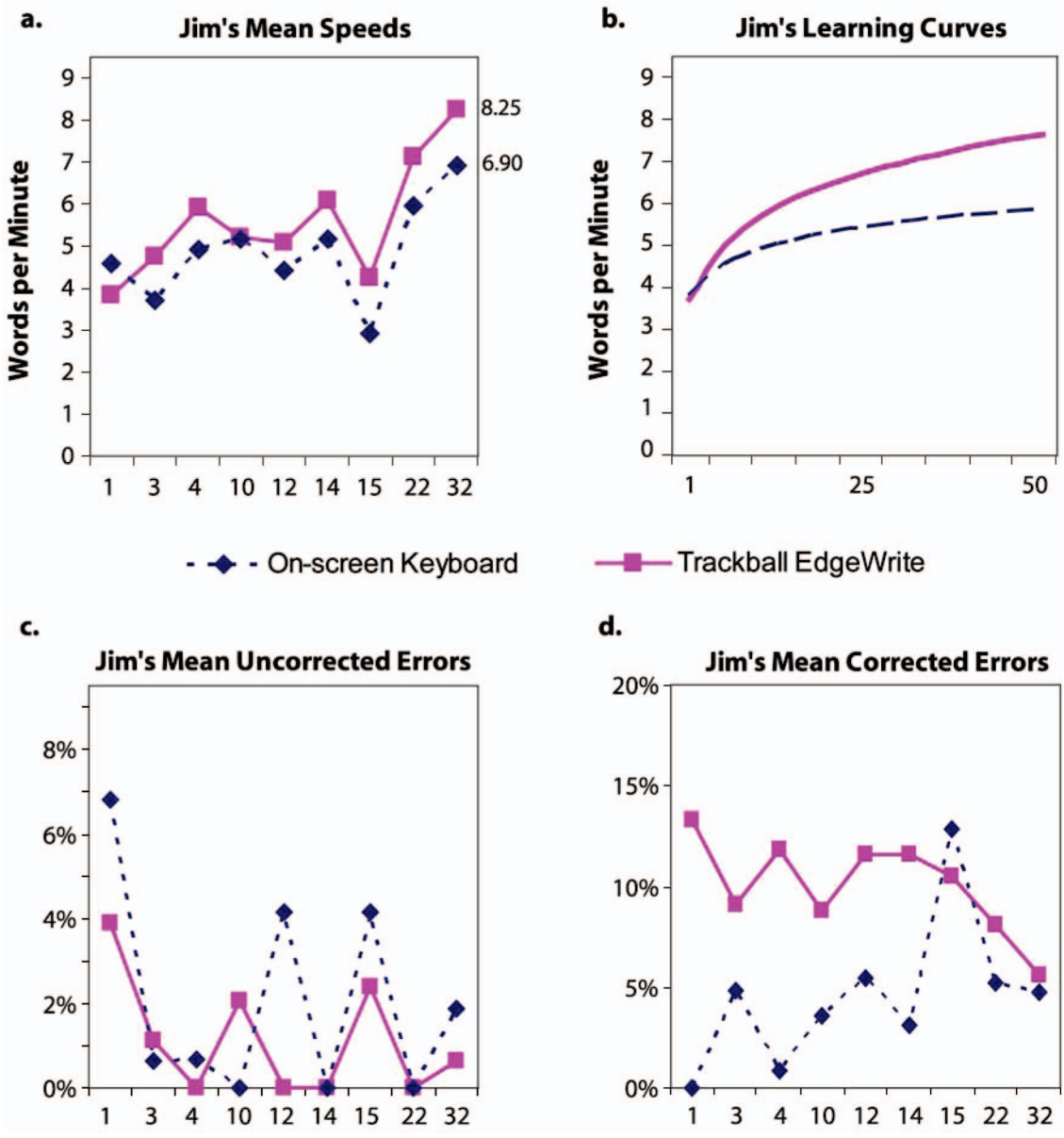

Figure 21. Jim's performance with his preferred on-screen keyboard and with *Trackball EdgeWrite*. The horizontal axis in each graph is the week number on which testing occurred.

methods were nearly equal in speed (Figure 21(*a*)). The second gap was simply a break in testing.

After week 1, Jim's speeds were consistently higher with EdgeWrite than with the MAK (Figure 21(*a*)). His average speed across all weeks with EdgeWrite was 5.61 wpm ($\sigma = 1.40$, max = 8.25). With the MAK it was 4.86 wpm ($\sigma = 1.17$, max = 6.90). A Wilcoxon sign test for matched pairs along weeks shows that his EdgeWrite speeds were significantly faster than the MAK's speeds ($z = -19.5$, $p < 0.02$).

The drop in performance during week 15 was due to a nagging tremor in Jim's hand. Interestingly, the drop in speed occurred about evenly for both methods. Accuracy results, conversely, show that both uncorrected and corrected errors[4] were worse for the MAK than for *Trackball EdgeWrite* (Figures 21(*c*) and (*d*)), suggesting that EdgeWrite accuracy may be less affected by such tremors.

Jim's speeds are modelled as learning curves and extended to week 50 in Figure 21(*b*). Although such an extension is speculative, it gives an idea of the possible trends for Jim's data. The power law equations and R^2 values for Jim's curves are:

$y = 3.713x^{0.1850}$ $R^2 = 0.53$ *Trackball EdgeWrite*
$y = 3.796x^{0.1127}$ $R^2 = 0.21$ The MAK

Jim's average uncorrected error rate was 1.12% ($\sigma = 1.39$) with EdgeWrite and 2.03% ($\sigma = 2.45$)

with the MAK (Figure 21(*c*)). Although the trend was in EdgeWrite's favour, a Wilcoxon sign test for matched pairs along weeks for uncorrected errors is not quite significant ($z = 8.0$, $p = 0.22$).

Jim's average corrected error rate was 10.06% ($\sigma = 2.37$) with EdgeWrite and 4.52% ($\sigma = 3.68$) with the MAK (Figure 21(*d*)). A Wilcoxon sign test for matched pairs along weeks for corrected errors yields a significant result in favour of the MAK ($z = -20.5$, $p < 0.02$). However, an analysis of the effect of *Session* on corrected errors indicates that it had a larger effect on decreasing corrected errors for EdgeWrite ($F_{1,7} = 5.21$, $p = 0.05$) than for the MAK ($F_{1,7} = 2.92$, $p = 0.13$). This indicates that corrected errors decreased significantly over time for EdgeWrite and presumably would continue to do so.

Although corrected errors – which are any letters backspaced during entry – are higher with *Trackball EdgeWrite*, they are not necessarily damaging if the correction operation is efficient [47]. Of course corrected errors should be reduced, but the main tradeoff is between speed and *uncorrected* errors, 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.

Jim no longer uses an on-screen keyboard, but keeps *Trackball EdgeWrite* running at all times, able to be called up at a moment's notice. To begin writing, Jim simply places the mouse cursor in the top-right corner of his screen, waits a moment and then watches as the cursor is 'captured' automatically within the EdgeWrite square (Figure 19). This notion of using the 'hot corners' of the desktop area to capture the mouse was Jim's idea. When Jim is done writing, he makes a dedicated stroke to 'release' his mouse cursor and dismiss the EdgeWrite window. The release stroke was also Jim's idea. Thus, Jim can operate *Trackball EdgeWrite* without ever clicking a mouse button (except to set preferences). Jim also helped to improve the feel of the software with suggestions for altering the on-screen visualization and stroke feedback.

Jim's reasons for preferring *Trackball EdgeWrite* over the MAK 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'.
- 'I feel like I can write again'.

These results show that *Trackball EdgeWrite* is faster and produces just as accurate, if not more accurate, text than an on-screen keyboard for Jim. Having reached a maximum speed of 8.25 wpm in week 32, it seemed unlikely that Jim could go much faster with the character-level version of *Trackball EdgeWrite*. After further innovation, word-level stroking was incorporated into the application (see Figure 16). What follows is a second-stage evaluation with Jim using word prediction and completion.

Word-level study. In order to empirically test word-level stroking in *Trackball EdgeWrite*, another evaluation was conducted with Jim. This evaluation compared word-level *Trackball EdgeWrite* to the on-screen keyboard *WiViK* (see Figure 3), which was configured with the prior settings from the MAK. WiViK was used instead of the MAK because of WiViK's powerful word prediction technology *WordQ*, which was loaded with the 'US Advanced' dictionary of 15 000 words. (The MAK, in contrast, lacks any word prediction feature.) This evaluation began $\sim$3 months after the end of the character-level study and $\sim$2 weeks after Jim had been introduced to word-level stroking in *Trackball EdgeWrite*.

The study was a single-subject 2-factor design, with factors for *Method* (WiViK, EdgeWrite) and *Word Prediction* (*WP*) (on, off). Jim did the word prediction versions second within both methods. A coin toss determined that he would use WiViK first. Thus, the technique order was: WiViK, EdgeWrite, WiViK + WP, EdgeWrite + WP. Jim entered three practice phrases and eight test phrases in each condition. Each phrase was $\sim$30 characters long.

Figure 22(*a*) shows Jim's speeds for the four conditions in the current study. It also shows Jim's peak speeds from the prior study (Figure 21(*a*)). Note the substantial speedups of both methods due to word prediction and completion. This indicates that the design for word-level stroking in *Trackball EdgeWrite* is beneficial to entry rates.

Figure 22(*b*) shows corresponding total error rates.[5] However, because Jim fixed almost all errors during entry, these total error rates are really just corrected error rates. Thus, *Trackball EdgeWrite* is producing similarly accurate text in a bit less time, albeit with more errors made (and fixed) along the way.

A Wilcoxon sign test for matched pairs for speed is not significant ($z = 3.0$, $p = 0.25$). However, the

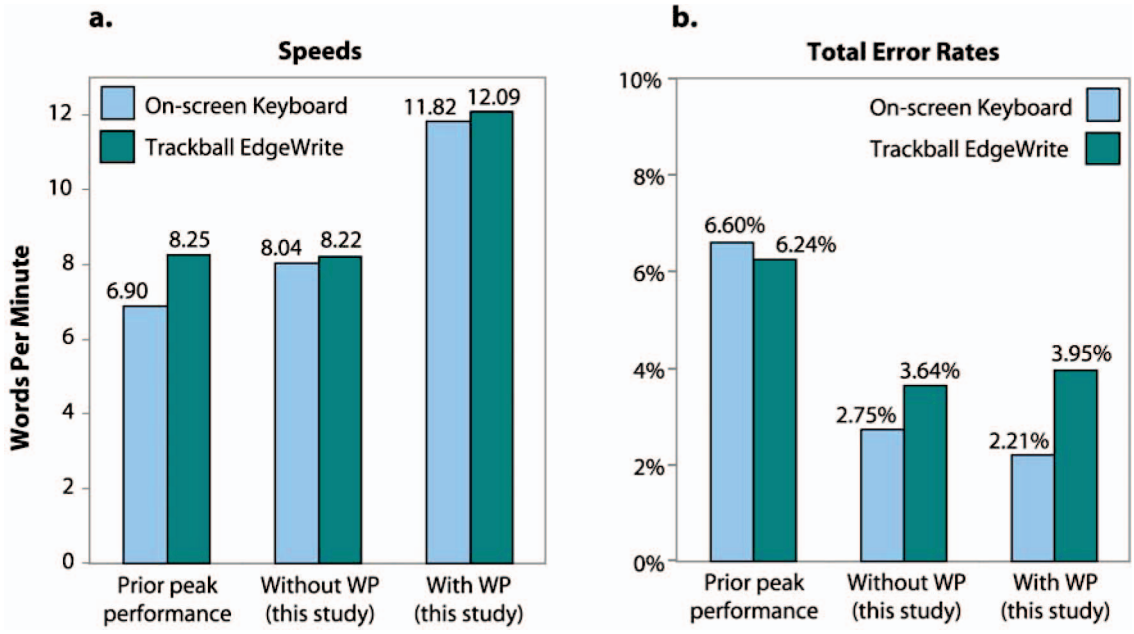

Figure 22. Jim's speeds and corresponding total error rates with an on-screen keyboard (WiViK) and word-level *Trackball EdgeWrite*.

trend is in favour of *Trackball EdgeWrite*. This advantage is only slight for the current study, however, probably because WiViK is a superior product to Jim's preferred keyboard (MAK), even if WiViK was configured with Jim's settings.

A Wilcoxon sign test for matched pairs for total errors is also not significant ($z = 2.0$, $p = 0.50$). However, both methods were producing error-free text in the end, since uncorrected errors for both methods were $\sim 0\%$.

It is interesting that character-level *Trackball EdgeWrite*'s errors were low in the current study even *without* word completion. This is probably because at the time of this second study, Jim had had more practice since his prior peak performance at the end of the first study.

It is worth noting that the speed of WiViK improved 32.0% with word prediction compared to without. This highlights the strength of WiViK's commercial word prediction and completion technology, WordQ.

Taken together, the results for word-level *Trackball EdgeWrite* show a 46.5% increase in speed and a 36.7% decrease in errors compared to Jim's prior peak performance with character-level *Trackball EdgeWrite*. The results also show that word-level *Trackball EdgeWrite* is 75.2% faster and 40.2% more accurate than Jim's prior peak performance with his own on-screen keyboard. Finally, the results show that word-level *Trackball EdgeWrite* is competitive with a major commercial product, the WiViK on-screen keyboard with WordQ word prediction.

Jim was asked to describe his experience with each of the four conditions in his own words. This is what he wrote:

- *WiViK*: '[Y]ou are constantly either scribbling around so you don't accidentally trigger the wrong letter, checking to see if you typed the right thing or looking for the next key to hover over. Too much work both mentally and visually'.
- *WiViK + WP*: 'Somewhat of a relief to hover over large words but it just increased the amount of mental and visual work required. [It's] one more section of the screen you need to scan constantly. Only thing is, I wish EdgeWrite had its vocabulary'.
- *EdgeWrite*: 'EdgeWrite without word prediction is like using a 286 or something. It's much better than a keyboard or an on-screen keyboard, but the ultimate is when you can flick the cursor into a corner and just pop the rest of the word in'.
- *EdgeWrite + WP*: 'The best thing about EdgeWrite is there is no eye strain or constant scanning between programs, letters, words, etc. The word choices are right there where your eyes already are. It actually helps you stay focused on what you're writing'.

Jim's sentiments confirm what prior studies of on-screen keyboards have found: that they are exceedingly tedious and visually intense [8]. Although word prediction improved WiViK's speed by 32.0%, it did

not resolve these drawbacks. *Trackball EdgeWrite*, on the other hand, proved to be just as fast but without the same visual tedium because it is gesture-based instead of selection-based, supporting writing 'by feel' rather than 'by sight'.

Log file analysis. As a third examination of Jim's performance, his text entry log files over 11 weeks of intermittent *Trackball EdgeWrite* use were analysed. With Jim's permission, all *Trackball EdgeWrite* text entry was recorded in XML files on his computer. Although such log files do not enable one to rigorously quantify speed and accuracy as in a controlled experiment, they do allow one to measure the stroke savings gained by using word prediction and completion. One can also look at the number of completions undone to gain some insight into selection accuracy and compare this to the number of letters undone (i.e. backspaced).

Figure 23 shows these quantities graphed over 2 months of Jim's intermittent use. It represents 897.52 hours of software running-time for 13 288 total strokes. Of these, 8774 were character strokes, 2201 were word-selection strokes, 1451 were backspaces and 249 were selection undos. In all, 15 629 characters were entered, 6855 of which were from completions. For example, if Jim stroked the letters 't' and 'h' and then selected 'there', this would result in six entered characters: two from character strokes and four from the selected completion. (The 6th character is a trailing automatic space.)

The top line shows the percentage of letters entered as predictions or completions. Without stroke-based word completion, these letters would all have to be entered in full. The weighted mean over all weeks is 43.9%. The spike in week 6 is an outlier due to a week of relative inactivity. Only 70 letters were entered that week, compared to most weeks which saw 1500–3500 letters. A regression line shows this trend to be slightly increasing.

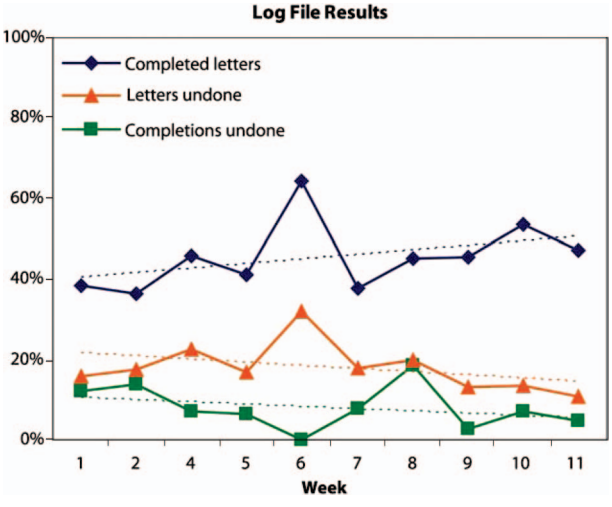

Figure 23. Results from 11 weeks of *Trackball EdgeWrite* use showing usage of word completion and backspace. Week 3 is omitted because Jim did not use his computer that week.

The bottom line is the percentage of word completions undone. The weighted mean over all weeks is 7.7%. As an indicator of completion errors, this value is probably inflated, since users may undo selected completions for reasons other than errors (e.g. as a result of changing what they want to write). A regression line shows this trend to be slightly decreasing.

For comparisons, the percentage of letters undone (backspaced) is shown as the middle line. The weighted mean for undone letters is 16.5%. This value is not surprising in light of previous results indicating that backspace is the second most common keystroke in desktop text entry [9]. A regression line shows this trend to be slightly decreasing.

Across all weeks, the average number of characters entered per completion was 3.11. Thus, with a simple 'pulse' into one of four corners, users avoid entering over three more characters for every single word they write.

Discussion

It is clear from the theoretical and empirical evaluations that word-level stroking improves the entry rate of *Trackball EdgeWrite* considerably. What is particularly interesting is that the *amount* of improvement indicated by these very different evaluations is quite similar percentage-wise. When taking into account an approximation for visual search time, the theoretical model showed a 45.0% increase in speed between the character-level and word-level versions of *Trackball EdgeWrite* – 23.1 vs 33.5 wpm. This theoretical difference is quite similar to the empirical 46.5% speedup obtained for Jim between his best character-level performance and his word-level performance (see Figure 22(*a*)). This is *also* similar to the 43.9% stroke savings obtained with word-level *Trackball EdgeWrite* as indicated by the log file analysis (see Figure 23). In other words, the theoretical prediction for a word-level advantage is actually manifested in quantifiable speed and stroke savings (i.e. efficiency improvements).

Furthermore, the fact that the efficiency improvements shown in the log files are similar to the measured speed improvement show that the stroke savings more or less translate directly to speed gains. This suggests that the perceptual, cognitive and motor costs of using *Trackball EdgeWrite*'s word-level stroking design are not overly taxing, which has been a problem with some prior word prediction systems [29,31].

Figure 24. An isometric joystick embedded in a mobile phone. The same software for *Trackball EdgeWrite* works for the isometric joystick, which can be controlled with the thumb.

It is clear from this work that the process of designing technology can benefit from incorporating both theoretical models and real-world participants. The use of a theoretical model helped correct an erroneous intuition in the design of *Trackball EdgeWrite*, namely that removing the need to move to the first corner would result in a faster technique (Figure 14). Although this intuition seemed straightforward, it turned out to be wrong both in theory and in practice.

Recent correspondence with Jim indicates that, 2 years after his initial exposure to *Trackball EdgeWrite*, he is still actively using the technology. He writes: 'I use EdgeWrite all time, especially if I am Googling things and I don't want to bother with the microphone and the dictation software. I think it's the handiest program ever written. . . . I have no complaints or major suggestions' (6 April 2007).

Future work

An obvious limitation of the current study is that it only used a single motor-impaired subject over multiple sessions. This was to promote focused and rapid iteration of the design. Thus, future work should include a study of more users and wider deployment for people with disabilities.

Although stroke-based word completion substantially increases the speed of *Trackball EdgeWrite*, it could still be improved upon. One of Jim's quotes indicated that he preferred the words offered by WiViK's WordQ to those offered by *Trackball EdgeWrite*. EdgeWrite's language models could be recreated using sources other than newspaper articles, perhaps including some of Jim's own texts. The culmination of this idea would be to provide the end-user with an interface to incorporate their *own* texts into EdgeWrite's language model.

The software for *Trackball EdgeWrite* may also be useful in an eye-tracking [25] or nose-pointing version [48]. Eye-tracking and nose-pointing can be used to move a mouse cursor and *Trackball EdgeWrite* would allow goal crossing to replace target pointing while writing. This may make it possible to write gesturally with one's eyes or nose.

The authors have also studied how *Trackball EdgeWrite* performs on an isometric joystick embedded in a mobile phone (Figure 24) [49,50]. Like trackballs, isometric joysticks have no notion of position, so the same software works without modification. Studies on mobile phones show that the character-level version of EdgeWrite is competitive with Multitap and that the word-level version is competitive with T9 (http://www.tegic.com), allowing able-bodied users to reach speeds above 16 wpm.

Conclusion

This article has presented *Trackball EdgeWrite*, a design for gesture-based text entry on everyday trackballs targeted at users with motor impairments. Among other design innovations, *Trackball EdgeWrite* includes a method for word-level stroking, which improves the speed and accuracy of the system over its character-level version without altering the way in which character strokes are made. In a study of a real trackball user with a spinal cord injury, his best prior performance with character-level *Trackball EdgeWrite* was both slower and less accurate than his performance with the new word-level version. The study also demonstrated that *Trackball EdgeWrite* rivals the major commercial on-screen keyboard

WiViK, being just as fast as WiViK but visually less tedious. Although *Trackball EdgeWrite* is more error prone *during* entry, it produces error-free text in the same amount of time due to efficient error correction. Over the course of both studies, the subject Jim went from a peak on-screen keyboard speed of 6.90 wpm without word prediction to 12.09 wpm with word-level *Trackball EdgeWrite*. His success with *Trackball EdgeWrite* even resulted in a television news story in which Jim was interviewed [51]. As it was for Jim, *Trackball EdgeWrite* could be useful to other motor-impaired users who wish to 'write' with their trackballs.

Acknowledgements

The authors thank Duen Horng Chau, John SanGiovanni, Richard Simpson 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 or any other supporter.

Portions of this article are adapted from [40,52].

Notes

1. 'Clutching' occurs when a user reaches the limit of allowable motion and must lift a device or finger to regain freedom to move in a particular direction. With a mouse, this happens when the mouse reaches the edge of the mouse pad or table. With a touchpad or trackball, it happens when the user's finger or thumb can no longer move in a given direction and therefore must be lifted.
2. This would turn out to work quite poorly, but the investigation was illuminating and so it is described here.
3. For example, 'ad' is a prefix for words 'added', 'addition', 'administration' and 'additional', all of which are more common than 'ad'.
4. During the test, Jim transcribes simple English phrases presented to him in both methods. Uncorrected errors are those that Jim leaves in his transcribed strings. Corrected errors are those that Jim initially makes but fixes (using backspace) during entry. See Soukoreff and McKenzie [47] for more details.
5. Total error rates are simply the sum of the uncorrected and corrected error rates.

References

1. Fuhrer CS, Fridie SE. There's a mouse out there for everyone. Proceedings of CSUN's 16th Annual Conference on Technology and Persons with Disabilities. Los Angeles, CA, 19–24 March, California State University Northridge; 2001.
2. Wu T-F, Wang H-P, Chen MC. Enabling computer access for children with cerebral palsy. Proceedings of the 11th International Conference on Human–Computer Interaction (HCI Int'l '05). Las Vegas, Nevada, 22–27 July. Mahwah, NJ: Lawrence Erlbaum Associates; 2005. On proceedings CD.
3. Dawe M. Complexity, cost and customization: Uncovering barriers to adoption of assistive technology. Refereed Poster at the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '04). Atlanta, Georgia; 18–20 October 2004.
4. Dawe M. Desperately seeking simplicity: How young adults with cognitive disabilities and their families adopt assistive technologies. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06). Montréal, Québec, 22–27 April. New York: ACM Press; 2006. pp 1143–1152.
5. Fichten CS, Barile M, Asuncion JV, Fossey ME. What government, agencies, and organizations can do to improve access to computers for postsecondary students with disabilities: Recommendations based on Canadian empirical data. Int J Rehabil Res 2000;23:191–199.
6. Guerette P, Sumi E. Integrating control of multiple assistive devices: A retrospective review. Assist Tech 1994;6:67–76.
7. Spaeth DM, Jones DK, Cooper RA. Universal control interface for people with disabilities. Saudi J Disabil Rehabil 1998;4:207–214.
8. Anson DK, Moist P, Przywara M, Wells H, Saylor H, Maxime H. The effects of word completion and word prediction on typing rates using on-screen keyboards. Proceedings of the RESNA 28th Annual Conference (RESNA '05). Atlanta, Georgia, 23–27 June. Arlington, Virginia: RESNA Press; 2005. On proceedings CD.
9. MacKenzie IS, Soukoreff RW. Text entry for mobile computing: Models and methods, theory and practice. Hum–Comp Interact 2002;17:147–198.
10. MacKenzie IS, Sellen A, Buxton W. A comparison of input devices in elemental pointing and dragging tasks. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '91). New Orleans, LO, March. New York: ACM Press; 1991. pp 161–166.
11. MacKenzie IS, Kauppinen T, Silfverberg M. Accuracy measures for evaluating computer pointing devices. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '01). Seattle, WA, 31 March–5 April. New York: ACM Press; 2001. pp 9–16.
12. Card SK, Mackinlay JD, Robertson G. The design space of input devices. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '90). Seattle, WA, 1–5 April. New York: ACM Press; 1990. pp 117–124.
13. Epps BW, Snyder HL, Muto WH. Comparison of six cursor devices on a target acquisition task. Proceedings of the Society for Information Display. San Diego, California, 6–8 May. San Jose, CA: Society for Information Display; 1986. pp 302–305.
14. Sperling BB, Tullis TS. Are you a better 'mouser' or 'trackballer'? A comparison of cursor-positioning performance. SIGCHI Bull 1988;19:77–81.
15. Accot J, Zhai S. Performance evaluation of input devices in trajectory-based tasks: An application of the Steering Law. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99). Pittsburgh, Pennsylvania, May 1999. New York: ACM Press; 1999. pp 466–472.
16. Paradise J, Trewin S, Keates S. Using pointing devices: Difficulties encountered and strategies employed. Proceedings of the 11th International Conference on Human–Computer Interaction (HCI Int'l '05). Las Vegas, Nevada, 22–27 July. Mahwah, NJ: Lawrence Erlbaum Associates; 2005. On proceedings CD.
17. Trewin S, Pain H. Keyboard and mouse errors due to motor disabilities. Int J Hum–Comp Stud 1999;50:109–144.
18. Hinckley K, Sinclair M. Touch-sensing input devices. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99). Pittsburgh, PA, 15–20 May. New York: ACM Press; 1999. pp 223–230.

19. Shein F, Hamann G, Brownlow N, Treviranus J, Milner M, Parnes P. WiViK: A visual keyboard for Windows 3.0. Proceedings of the RESNA 14th Annual Conference (RESNA '91). Kansas City, MI, 21–26 June. Washington, DC: RESNA Press; 1991. pp 160–162.
20. Bishop JB, Myers GA. Development of an effective computer interface for persons with mobility impairment. Proceedings of the 15th Annual Conference on Engineering in Medicine and Biology. Los Alamitos, CA: IEEE Press; 1993. pp 1266–1267.
21. Ward DJ, Blackwell AF, MacKay DJC. Dasher – A data entry interface using continuous gestures and language models. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '00). San Diego, CA, 6–8 November. New York: ACM Press; 2000. pp 129–137.
22. Ward DJ, MacKay DJC. Fast hands-free writing by gaze direction. Nature 2002;418:838.
23. Isokoski P, Raisamo R. Device independent text input: A rationale and an example. Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI '00). Palermo, Italy, May. New York: ACM Press; 2000. pp 76–83.
24. Wobbrock JO, Myers BA. Gestural text entry on multiple devices. Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '05). Baltimore, MD, 9–12 October. New York: ACM Press; 2005. pp 184–185.
25. Isokoski P. Text input methods for eye trackers using off-screen targets. Proceedings of the ACM Symposium on Eye Tracking Research and Applications (ETRA '00). Palm Beach Gardens, FL, November. New York: ACM Press; 2000. pp 15–21.
26. Zhai S, Kristensson P. Shorthand writing on stylus keyboard. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03). Ft. Lauderdale, FL, 5–10 April. New York: ACM Press; 2003. pp 97–104.
27. Mankoff J, Abowd GD. Cirrin: A word-level unistroke keyboard for pen input. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98). San Francisco, CA, November. New York: ACM Press; 1998. pp 213–214.
28. Perlin K. Quikwriting: Continuous stylus-based text entry. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98). San Francisco, CA, 1–4 November). New York: ACM Press; 1998. pp 215–216.
29. Goodenough-Trepagnier C, Rosen MJ, Galdieri B. Word menu reduces communication rate. Proceedings of the RESNA 9th Annual Conference (RESNA '86). Minneapolis, MN, 23–26 June. Washington, DC: RESNA Press; 1986. pp 354–356.
30. Koester HH, Levine SP. Effect of a word prediction feature on user performance. Augment Alt Comm 1996;12:155–168.
31. Soede M, Foulds RA. Dilemma of prediction in communication aids. Proceedings of the RESNA 9th Annual Conference (RESNA '86). Minneapolis, Minnesota, 23–26 June. Washington, DC: RESNA Press; 1986. pp 357–359.
32. Wobbrock JO, Myers BA, Kembel JA. EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03). Vancouver, British Columbia, 2–5 November. New York: ACM Press; 2003. pp 61–70.
33. Fitts PM. The information capacity of the human motor system in controlling the amplitude of movement. J Exp Psychol 1954;47:381–391.
34. MacKenzie IS. Fitts' law as a research and design tool in human–computer interaction. Hum–Comp Interact 1992;7: 91–139.
35. Accot J, Zhai S. Beyond Fitts' law: Models for trajectory-based HCI tasks. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97). Atlanta, Georgia, 22–27 March. New York: ACM Press; 1997. pp 295–302.
36. Accot J, Zhai S. More than dotting the i's: Foundations for crossing-based interfaces. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '02). Minneapolis, Minnesota, 20–25 April 2002. New York: ACM Press; 2002. pp 73–80.
37. Kristensson P, Zhai S. SHARK^2: A large vocabulary shorthand writing system for pen-based computers. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '04). Santa Fe, New Mexico, 24–27 October. New York: ACM Press; 2004. pp 43–52.
38. Wobbrock JO, Myers BA, Chau DH. In-stroke word completion. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '06). Montreux, Switzerland, 15–18 October. New York: ACM Press; 2006. pp 333–336.
39. Zipf G. Selective studies and the principle of relative frequency in language, Cambridge, MA: MIT Press; 1932.
40. Wobbrock JO, Myers BA. Trackball text entry for people with motor impairments. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06). Montréal, Québec, 22–27 April. New York: ACM Press; 2006. pp 479–488.
41. Kucera H, Francis WN. Computational analysis of present-day American English, Providence, RI: Brown University Press; 1967.
42. Clarkson PR, Rosenfeld R. Statistical language modeling using the CMU-Cambridge toolkit. Fifth European Conference on Speech Communication and Technology (Eurospeech '97). Rhodes, Greece; September 1997. pp 2707–2710.
43. Hick WE. On the rate of gain of information. Q J Exp Psychol 1952;4:11–26.
44. Hyman R. Stimulus information as a determinant of reaction time. J Exp Psychol 1953;45:188–196.
45. Soukoreff RW, MacKenzie IS. Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard. Behav Inf Tech 1995;14:370–379.
46. Horstmann HM, Levine SP. The effectiveness of word prediction. Proceedings of the RESNA 14th Annual Conference (RESNA '91). Kansas City, MI. 21–26 June. Washington, DC: RESNA Press; 1991. pp 100–102.
47. Soukoreff RW, MacKenzie IS. Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03). Ft. Lauderdale, FL, 5–10 April. New York: ACM Press; 2003. pp 113–120.
48. Austen I. A new rule of cursor control: Just follow your nose. New York: The New York Times; 28 October 2004.
49. Chau DH, Wobbrock JO, Myers BA, Rothrock B. Integrating isometric joysticks into mobile phones for text entry. Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '06). Montréal, Québec, 22–27 April. New York: ACM Press; 2006. pp 640–645.
50. Wobbrock JO, Chau DH, Myers BA. An alternative to push, press, and tap-tap-tap: Gesturing on an isometric joystick for mobile phone text entry. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '07). San Jose, CA, 28 April–3 May. New York: ACM Press; 2007. pp 667–676.
51. Ivanhoe Broadcast News. Hi-tech typing. Discoveries and breakthroughs inside science. Winter Park, FL, USA: The American Institute of Physics; 2005.
52. Wobbrock JO, Myers BA. From letters to words: Efficient stroke-based word completion for trackball text entry. Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '06). Portland, OR, 23–25 October. New York: ACM Press; 2006. pp 2–9.