Performance Optimizations of Virtual Keyboards for Stroke-Based Text Entry on a Touch-Based Tabletop

Jochen Rick
Department of Computing
The Open University
Milton Keynes, MK7 6AA, UK
j.rick@open.ac.uk

ABSTRACT

Efficiently entering text on interactive surfaces, such as touch-based tabletops, is an important concern. One novel solution is *shape writing*—the user strokes through all the letters in the word on a virtual keyboard without lifting his or her finger. While this technique can be used with any keyboard layout, the layout does impact the expected performance. In this paper, I investigate the influence of keyboard layout on expert text-entry performance for stroke-based text entry. Based on empirical data, I create a model of stroking through a series of points based on Fitts's law. I then use that model to evaluate various keyboard layouts for both tapping and stroking input. While the stroke-based technique seems promising by itself (i.e., there is a predicted gain of 17.3% for a Qwerty layout), significant additional gains can be made by using a more-suitable keyboard layout (e.g., the OPTI II layout is predicted to be 29.5% faster than Qwerty).

Author Keywords

Keyboard layout, virtual keyboard, touch input, interactive tabletops, shape writing, Fitts's law

ACM Classification Keywords

H5.2 [Information interfaces and presentation]: User Interfaces. - Input devices and strategies.

General Terms

Design, Human Factors

TEXT ENTRY ON TOUCH-BASED TABLETOPS

In a typical desktop setup, text is entered efficiently and comfortably using a physical keyboard. Unfortunately, the same setup is impractical in other computing contexts. For handheld devices, a full-size keyboard is unwieldy. Because of their popularity, much academic research and commercial development has gone into creating text-entry solutions for these devices [18]. While there are novel ways of text entry without a keyboard (e.g., speech, handwriting, and sign-language recognition [12]), there has been success in adapting keyboards to the smaller form factor [18]. Scaled-down physical keyboards allow users to apply their familiarity with the Qwerty layout to the new interface, even if ten-finger typing is prohibited by the minuscule key size. Another popular option is to use a virtual keyboard on a stylus- or touch-sensitive display. Virtual keyboards can be remapped easily (e.g., switching between entering letters and numerals) and can be hidden when not in use, thereby conserving space.

Another computing platform where a physical keyboard may not be the optimum solution is an *interactive tabletop*—a large horizontal surface that responds to touch input. While tabletops are large enough to accommodate a full-size keyboard, the presence of a physical keyboard reduces the access the user has to the interactive surface. Since tabletops are typically large enough where access is already a serious concern, using physical keyboards is impractical. Additionally, tabletop users are often standing to increase reach, making it awkward to use a ten-finger typing technique. While applications designed for tabletops tend not to rely heavily on text entry, it is a serious concern [9, 25]. Yet, the design space for tabletop text entry remains largely unexplored [10].

For tabletops, virtual keyboards are a good option, since they can pop-up when necessary to conserve space. For several reasons, it is appropriate to use a much larger virtual keyboard on a tabletop than on a handheld device. First, the distance between the user's eye and the interactive surface is substantially larger. Second, the input resolution of tabletops is usually low enough to require a sizable distance between keys. Third, tabletop input is generally through using fingers, which are fat compared to the styli that are still common on handheld devices. While it may make sense to create a virtual keyboard that shares the same size as a physical keyboard, ten-finger typing is still problematic. For technologies where pressure cannot be sensed, a user must hover above the keyboard area, rather than lightly resting on the keys, as is typical with a physical keyboards. Even if pressure can be sensed, there is still no tactile feedback to let the user know when a key tap has been registered.

One interesting alternative to tap-based virtual keyboards, where the letters in the word are tapped sequentially, is to use *stroke-based virtual keyboards*, where the user drags one finger sequentially through the letters of the word in a single gesture [19]. In the case of ambiguity, a lexicon-based algorithm can usually map the stroke to the intended word

UIST'10, October 3–6, 2010, New York, New York, USA..

[13]. For stroke-based entry, the lack of tactile-feedback is less of a problem and there is only the need to use one finger, which suits both technologies that would register any touch and contexts where the user is not in a comfortable position to use all ten fingers.

While stroke-based text entry can be performed with a Qwerty layout, the distance between consecutive letters in a word (see Figure 3) tends to be long and the stroke shape involves large horizontal movements. A more suitable keyboard layout could increase efficiency and decrease ambiguity to increase the value of the stroke-based technique. In this paper, I introduce a model for stroke movement through a series of points. I use that model to evaluate established keyboard layouts and generate two keyboard layouts which are more optimized for stroke-based text entry than existing layouts. To introduce the issues involved in optimizing a keyboard and to introduce the ideas behind stroke-based text entry, I start with a brief history of keyboard layouts.

A HISTORY OF KEYBOARD LAYOUTS

This section introduces a number of different keyboard layouts in chronological order; they are also displayed in chronological order in Figure 1. The purpose of introducing these layouts is both to mention related work and to introduce layouts of interest which will later be evaluated for their upper bound in text entry speed.

Keyboards originated with the invention of the typewriter [26]. For early models, just getting the mechanics to work correctly was a significant challenge. Key presses required serious effort, so ten-finger typing was not even considered a possibility. Sholes invented the Qwerty layout (Figure 1a) in 1874. He initially based his layout on the alphabet. Remnants of this can still be seen in the home row, which almost contains the sequence of letters from A to L. An immediate problem he ran into was that keys close to each other would often jam when pressed in rapid succession. To remedy this, he altered the layout so that keys that bordered each other would not often be hit in rapid succession. The shape of the keys also had to be staggered to allow for maximum force transmittal. His Qwerty layout worked and soon became the dominant keyboard layout for typing English. While Sholes's constraints for a good keyboard layout are no longer valid in our computerized world, his keyboard layout remains.

The most notable attempt to improve on Qwerty was done in 1936 by Dvorak and Dealey. The Dvorak simplified keyboard (Figure 1b) optimized the typewriter layout for ten-finger typing based on a *digraph table*, a chart mapping the transition probability between letters, for typical English. To maintain compatibility with the Qwerty-dominated typewriter market, the shape (three wide staggered rows) was maintained and only the key mapping was changed. The layout was changed so that the most common letters were placed in the home row and the layout encouraged alternating letters from one hand to the other. Dvorak is easier to learn, more accurate, faster to work with, and less fatiguing than Qwerty [28]. While the Dvorak keyboard still maintains a small following, it failed as a business venture [26]. Keyboards based on the alphabet (Figure 1c) were also marketed, but these were found not to be terribly advantageous once users gain familiarity with the keyboard. One possible reason for this is that a simple visual search for the next key might actually be faster and less-cognitively taxing than trying to determine the position based on recall of the alphabet [21].

In 1980, the Montgomery wiping keyboard (Figure 1d) was introduced as the first stroke-based keyboard [20]. A user could stroke through a series of keys on a touch-sensitive surface and each one would be transmitted once to the computer. The layout was designed to maximize the length of a stroke. One of the more unusual design choices is to provide two keys for each of the letters I, O, and N.

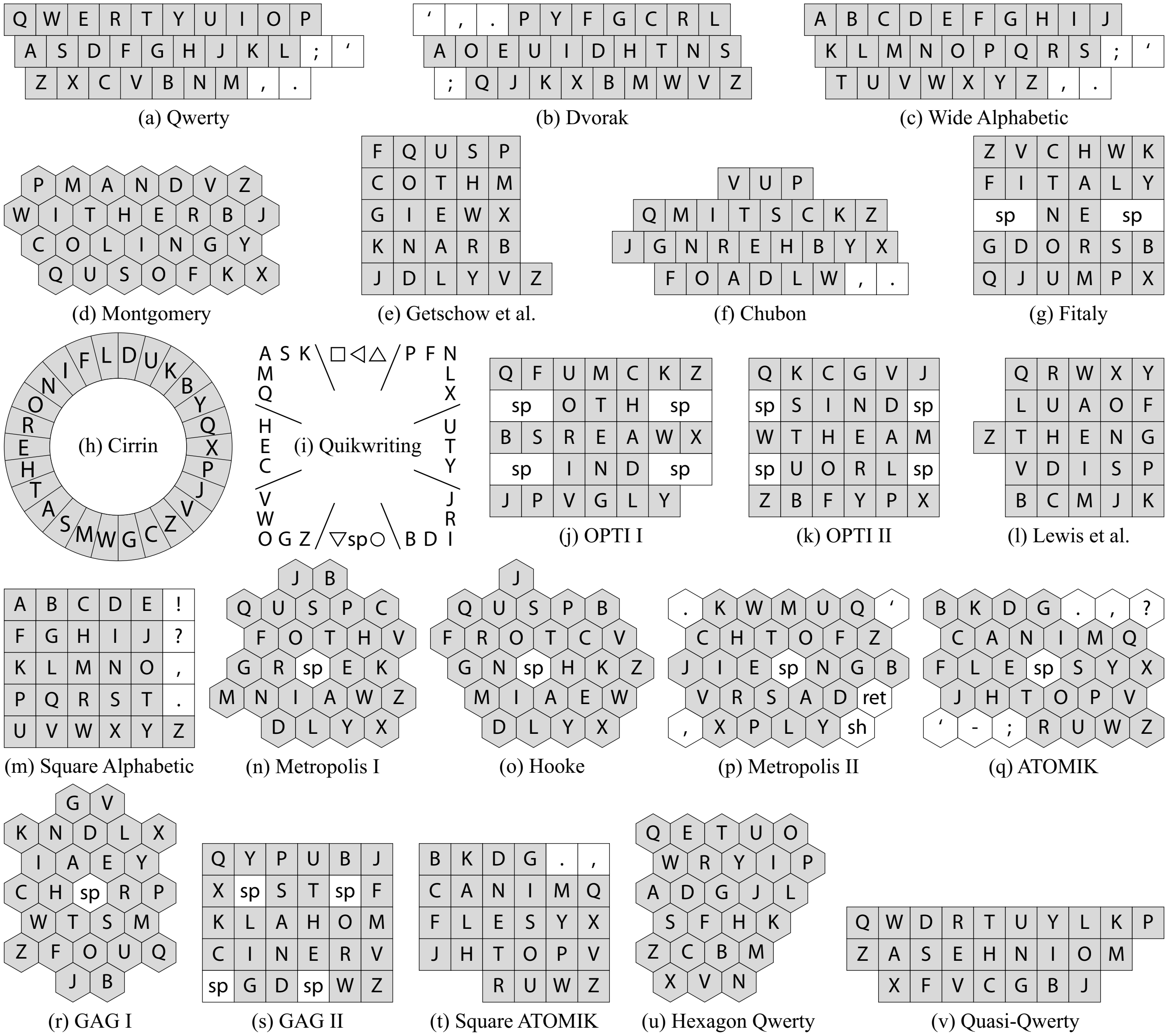
As other computational devices became available, different keyboard layouts were created to allow for better input rates for one finger typing. The layout introduced by Getshow et al. (Figure 1e) was optimized based on a greedy algorithm of key frequency [8]. The Chubon layout (Figure 1f) modified an existing physical keyboard (hence the four rows) to better support disabled users who could only use a single point of contact [6]. The Fitaly keyboard[1] (Figure 1g) is a commercial product available for devices that use one point of contact (e.g., a stylus or finger) [27]; it was particularly marketed for its use for personal digital assistants (PDAs), which were gaining commercial acceptance at that time.

Cirrin (Figure 1h), short for "circular input," is the first word-level stroke-based keyboard. A user transitions through all the letters in order. To avoid ambiguity, the user transitions through the empty center to avoid hitting unwanted keys [19]. Introduced at the same conference, Quikwriting (Figure 1i) allows users to use stroke input to enter a series of characters [23]. A letter is entered for each return to the center region and based on the first region entered and the last region exited before returning to the center region.

When handheld computers became more popular, increased research interest was raised in creating better keyboard layouts for one input, whether a finger or a stylus. OPTI I [17] and OPTI II [36] (Figures 1j & 1k) were created based on heuristics of the English language. Lewis et al. [14] created another compact optimized layout (Figure 1l); they also created an alphabetic one (Figure 1m) to compare how novice users responded to these layouts. In a user study, the tall alphabetic layout was preferred by novice users over more optimized layouts [15]; however, a simple Qwerty layout was preferred significantly more than any of the compact layouts.

Algorithmic attempts to optimize a keyboard based on Fitts's law followed. A group at IBM created several layouts using simulated annealing algorithms [11, 29]. The Hooke layout (Figure 1o) was created using a digraph model where keys are linked to each other through elastic springs; the layout was named after Hooke's law. The Metropolis I and II layouts (Figure 1n & 1p) were created with a Metropolis random walk algorithm. The same group then produced the ATOMIK layout (Figure 1q) to introduce an alphabetic bias (with the alphabet tending to be spread from top left to bottom right) into the Metropolis algorithm; while it was predicted to be

[1] © Textware Solutions 1996–2009



(a) Qwerty (b) Dvorak (c) Wide Alphabetic

(d) Montgomery (e) Getschow et al. (f) Chubon (g) Fitaly

(j) OPTI I (k) OPTI II (l) Lewis et al.

(m) Square Alphabetic (n) Metropolis I (o) Hooke (p) Metropolis II (q) ATOMIK

(r) GAG I (s) GAG II (t) Square ATOMIK (u) Hexagon Qwerty (v) Quasi-Qwerty

Figure 1: Various Keyboard Layouts

slightly slower than Metropolis II for expert users, novices were more successful with ATOMIK [30]. The GAG I (Figure 1r) and GAG II (Figure 1s) layouts were created to improve on the hexagonal layout of the Metropolis II keyboard and rectangular layout of OPTI II using a genetic algorithm based on a digraph table [24].

Shape writing was introduced as an input technique that could match the shape of a stroke to its corresponding word [32]. Unlike Cirrin, a shape-writing keyboard has to maintain a lexicon of words and the mapping between strokes and those words. Initially, shape writing was meant to be used for only a few common words as the algorithm did not scale to a larger lexicon [31]. A second algorithm was created that allowed for large-lexicon shape writing [13]. While users of shape writing are instructed to stroke through the letters of a word, the recognition engine in both cases [13, 31] is based on matching the shape of the stroke rather than its absolute position. As such, the term “shape writing” encapsulates both a novel input technique (word-level stroke-based input) and a mechanism for mapping input to output (recognizing a shape independent of position). As the latter can be accomplished in multiple ways (e.g., I have developed an algorithm for recognizing words based on absolute position), I will use the term “stroke-based input” in this paper to separate the technique from the recognition mechanism. As shape writing is a whole-word technique, there was no great need for a space key to be easily accessible. Zhai and Kristensson [32] created a square-key version of the ATOMIK layout (Figure 1q) without the central space key. This layout is provided as an option on the commercially-available ShapeWriter software.

As has been shown, novice users prefer a Qwerty layout as they are familiar with it. So, it might make sense to create a new layout that is similar to Qwerty. One version (Figure 1u) staggers the keys into separate rows to change both the shape of the keyboard (from wide to compact) and to lower the amount of ambiguity when shape writing [33]. Quasi-Qwerty (Figure 1v) is an digraph optimized keyboard where each letter is at most one key away from where it would be on the Qwerty layout; this facilitates easier visual search for users familiar with Qwerty [3].

On a simple level, stroke-based text entry shares criteria with tap-based text entry. The longer the distances between letters, the longer it takes to reach them with either tapping or stroking. So, a compact layout with letters that are often in succession next to each other is advantageous to both techniques. On the other hand, stroking is faster when drawing a straight line through some points rather than zigzagging; for tapping, as long as the distances between the keys is the same, it makes little difference as each tap requires a full stop to a movement. So, a slightly different model is necessary to model stroke-based text entry.

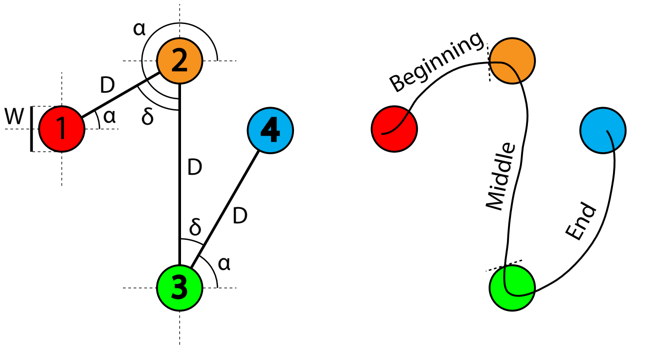
CREATING A MODEL

Tapping interfaces have been shown to closely follow Fitts's law. Thus, developers have been able to evaluate or generate keyboard layouts using that simple mathematical model and a digraph table. While some works expands on Fitts's law to model more sophisticated actions, such as drawing a spiral [1], there is no model for stroking through a number of points with turns [13]. The closest match is the work of Cao and Zhai [4] on modeling arbitrary pen movements. The premise of that work is that a shape can be broken down into a combination of straight lines, smooth curves, and corners and that the time needed to draw the shape is a sum of these components. For stroke-based input, the model would be a series of lines separated by corners. While that is a reasonable first-pass approximation, cutting through corners can significantly decrease the time as multiple parts are combined into a single chunk. Chunking behavior is a particular weakness of the Cao and Zhai model as the model predictions consistently overestimate the time a gesture will take [4, 5]. Chunking is the crux of the problem for modeling stroking as people are good at cutting corners to optimize performance [22] . To create a better approximation, I conducted an empirical study to investigate the role of distance and angle on the different segments of a stroke sequence.

A User Study

Participants completed a series of stroke sequences through four points, in order from 1 to 4 (Figure 2a). A four-point sequence was chosen so that each stroke sequence included a beginning (beginning a stroke), a middle (continuing an existing stroke), and an end (completing the stroke by stopping at the end point and lifting the finger).

For each stroke sequence in the series, the distance (D) was varied by an integer multiple of the diameter of a point (W) from 1 to 8. The angles (α and δ) were varied as multiples of 30° from 0° to 330°. The middle and end sections were counterbalanced so that each participant would complete one sequence for each combination of direction angle (α), change of direction angle (δ), and distance ratio ($\frac{D}{W}$). This necessitated 1152 sequences per participant (12 choices for α multiplied by 12 choices for δ multiplied by 8 choices for $\frac{D}{W}$). The beginning path was counterbalanced to evenly present each combination of direction angle and distance ratio (i.e., 12 instances of each combination per participant). Given those criteria, beginning, middle and end sections were then randomly combined and shuffled for each participant.



(a) Input Parameters (b) Output Times

Figure 2: One Three-Part Stroke Sequence

Eight adults (6 right-handed, 2 left-handed) participated in the study. All had previous experience using the DiamondTouch table [7] used for the study. The target points were projected onto the top of the table and sized so that the diameter (W) of a point corresponded to a typical spacing between keys on a physical keyboard (2.1 cm). Participants were informed that they should aim to complete a stroke sequence as accurately and quickly as possible. They were also informed that they did not have to stop at points 2 and 3, but could just stroke through them. The software only accepted a sequence if it started at 1, stroked through 2 and 3, and ended at 4; otherwise, the participant would have to attempt the stroke sequence again. While numbering and coloring the points worked well to indicate order, occasionally participants would make mistakes. To correct for that, participants were able to back up and redo a stroke sequence that they felt did not match the criteria of accurate and quick. Participants stood during the task to increase their access to the table. The stroke sequences were placed near their table position to assure that the entire stroke sequence would be within their region of comfortable reach. Each participant completed a training task of over 100 sequences to become familiar with the task. As this was a lengthy and monotonous task, participants were encouraged to take breaks. The sessions typically lasted about one hour.

As the sampling rate of the DiamondTouch is only around 20Hz, the procedure to determine whether a point had been stroked was to see whether a line connecting the current touch location to the previous touch location intersected the point. A similar procedure based on the fraction of the line within the point was used to determine how much time was spent on the $Beginning$, $Middle$, and End segments (Figure 2b).

The main study provides data to model stroke input for more than two points. To model stroke sequences of two points and tapping, a small follow-up study was conducted with

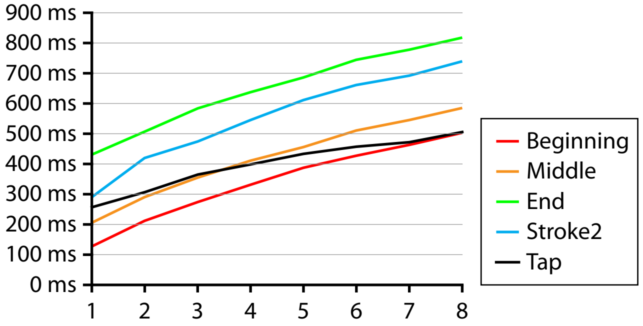


Figure 3: Stroke Time by Distance Ratio

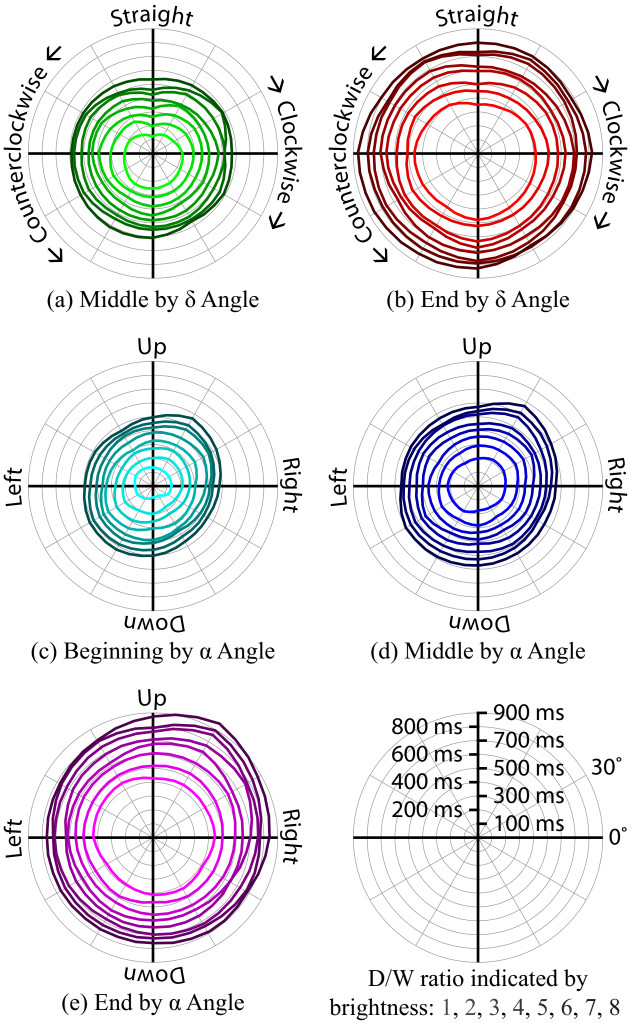
six users, with a large overlap with the previous participants. Each participant completed 96 sequences for both tapping and stroking between two points (12 choices for α multiplied by 8 choices for $\frac{D}{W}$).

Study Results

Figure 3 plots the averages of the stroke times by the distance ratio. One of the principles behind Fitts's law is that similar actions should conform to a similar mathematical model [16]. So, it is not surprising that the effect of the distance ratio for $Beginning$, $Middle$, and End seems to be identical, with the three curves aligning closely when shifted vertically. In contrast, the curve for $Tapping$ is a different shape; this is expected as moving a finger through the air is substantially different than dragging a finger across a surface. As End involves slowing down to come to a stop at point 4, it takes longer than $Middle$; the time to slow down seems to be unaffected by the distance ratio.

When the effect of the directional angle α was examined for different participants, there was a noticeable and similar pattern, except for the two left-handed participants. When the left-handed participants' data was flipped across a vertical line of symmetry, the pattern then matched the right-handed participants. To better model the effect of angle, the data for the left-handed participants was flipped to mirror their right-handed counterparts.

Figure 4 plots the effect that α and δ had on the timing based on the eight different distance ratios. For both $Middle$ (Figure 4a) and End (Figure 4b), the δ curves resemble an apple offset down from the center. What this shape indicates is that it takes the least time to continue straight and the most time to change direction by 180°. For $Beginning$ (Figure 4c), $Middle$ (Figure 4d), and End (Figure 4e), the δ curves resemble an olive tilted at an angle. What this shape indicates is that it is takes longer to stroke down-left and up-right than it does to stroke down-right and up-left. A possible explanation for this shape is that this is a function of arm direction: The more efficient angles are when the movement direction is parallel to the arm and the less efficient angles are when the movement direction is orthogonal to the arm. One concern that colleagues had going into the study was that there would be a significant inefficiency for the down-right direction as that is when the arm is visually blocking the next point. This did not seem to be the case.



(a) Middle by δ Angle

(b) End by δ Angle

(c) Beginning by α Angle

(d) Middle by α Angle

(e) End by α Angle

D/W ratio indicated by brightness: 1, 2, 3, 4, 5, 6, 7, 8

Figure 4: Stroke Time by Angle

Creating a Mathematical Model

Modeling movement mathematically has traditionally been done using Fitts's law, which states that the amount of time an activity takes maps to a logarithmic function of the distance over target width ratio:

1. $Time = a + b\,log_2(\frac{D}{W} + c)$

While a and b are variables which are adjusted to fit the data, there are three common variants for c [16]. In Fitts's initial formulation, $c = 0$. In Welford's variation, $c = 0.5$. In a direct correlation to Shannon's information theorem, $c = 1$. Both tapping between points and simple stroking through a border have been successfully modeled using the Shannon formulation Fitts's law [2]. In this case, directly applying a Fitts's law formulation is problematic as the paths, demonstrated in Figure 2b, are significantly longer than the distance between points. To compensate for this additional distance, I allow c to be determined by the input data. Using this variation on Fitts's law, I solved for a, b, and c to best match the different segments (Figure 3). Solving independently for

$Beginning$, $Middle$, and End led to similar values for b $(262.8, 291.1,$ and $305.9)$ and c $(3.12, 3.77,$ and $3.97)$. This variance is further reduced if only one of b and c is allowed to vary. As b values are hypothesized to be the same for similar tasks, I averaged the three curves to arrive at a single b_S and a single c_S. I then used those values to solve for a_{SB}, a_{SM}, and a_{SE}. To gage the validity of this approach, I compared this model to one based on a Shannon formulation of Fitts's law (i.e., $c = 1$) and found that R^2 (the coefficient of determination) increased from 0.992 to 0.999 with the new formulation.

To adjust for the effect of angles, I create two functions (F_α and F_δ) to adjust these values. Since the effect of the α angle (Figures 4c, 4d, & 4e) stretches the shapes fairly evenly in regards to the distance ratio, I model F_α as a multiplicative adjustment. Since the three shapes in Figures 4c, 4d, & 4e are similar, I combine them to arrive at one F_α. This adjustment increased R^2 from 0.955 to 0.978. Since the effect of the δ angle (Figures 4a & 4b) seems to be concentrated at the center, with each curve based on a higher distance ratio radiating outwards from that apple-shaped core, I model F_δ as an additive adjustment. This is the same approach used by Cao and Zhai [4], who model corners as a function of the δ angle. As the effect of the δ angle appears to be the same for both $Middle$ and End, I combine the data. As the direction of the turn does not greatly impact the result (i.e., both Figures 4a & 4b appear to have a vertical line of symmetry), the data was further simplified to average clockwise and counterclockwise values to arrive at one F_δ. This adjustment increased R^2 from 0.933 to 0.984. This leads to the following three equations:

2. $Beginning = (a_{SB} + b_S log_2(\frac{D}{W} + c_S))F_\alpha$

3. $Middle = (a_{SM} + b_S log_2(\frac{D}{W} + c_S))F_\alpha + F_\delta$

4. $End = (a_{SE} + b_S log_2(\frac{D}{W} + c_S))F_\alpha + F_\delta$

Using these three equations, an arbitrary stroke of three or more points can be modeled as a combination of one *Beginning* stroke, a variable number of $Middle$ strokes, and one End stroke. For the case of stroking between two points, there is no reason to adjust c. Previous work [2] has shown that the Shannon formulation ($c = 1$) is appropriate for this task. I therefore matched the data ($Stroke2$ in Figure 3) to determine values for a_{S2} and b_{S2}. As the angle of the stroke should still impact the timing, I retain F_α in the model:

5. $Stroke2 = (a_{S2} + b_{S2} log_2(\frac{D}{W} + 1))F_\alpha$

To model tapping, a Fitts's law equation is used to match the Tap data in Figure 3 to a_T and b_T. Again, previous studies were able to successfully match tapping data using the Shannon formulation of Fitts's law [16]. Unfortunately, the tapping data set is too small to investigate the effect of angle. As lifting a finger is a substantially different action than dragging, it cannot be assumed that the same F_α will apply. Instead, no adjustment is made for angle in the tapping model. As the focus of this work is on modeling stroking, this does not majorly impact the findings.

Stroking		**Tapping**	
Constant	*Value*	*Constant*	*Value*
a_{SB}	-507.34 ms	a_T	128.90 ms
a_{SM}	-428.12 ms	a_{TJ}	49.03 ms
a_{SE}	-199.63 ms	b_T	114.88 ms
b_S	285.31 ms		
c_S	3.59		
a_{S2}	-21.22 ms		
b_{S2}	246.97 ms		

Table 1: Best Fitting Constants

$angle$	$F_\alpha(angle)$	$F_\delta(angle)$
0°	98.49%	-69.19 ms
30°	104.86%	-28.77 ms
60°	109.01%	-7.49 ms
90°	100.94%	11.95 ms
120°	98.42%	17.33 ms
150°	98.99%	20.97 ms
180°	99.20%	41.22 ms
210°	102.42%	20.97 ms
240°	100.18%	17.33 ms
270°	96.97%	11.95 ms
300°	94.94%	-7.49 ms
330°	95.59%	-28.77 ms

Table 2: Sampled Values for F_α and F_δ

6. $Tap = a_T + b_T log_2(\frac{D}{W} + 1)$

7. $TapJump = a_{TJ} + b_T log_2(\frac{D}{W} + 1)$

With this formulation, an arbitrary tap sequence can be modeled as combination of its components taps (Tap). For modeling stroke-based interaction, the tapping model is also necessary to assess the time between strokes ($TapJump$). Based on matching the empirical data, Table 1 shows the calculated values of the constants and Table 2 shows the calculated values for the angle functions for the tested angles. When applying the model, the values of F_α and F_δ are linearly interpreted between the measured values. One striking feature of the calculated values are the large negative values for a in stroking. This is a direct consequence of letting c vary, as the value of the logarithmic component is substantial even for low values of the $\frac{D}{W}$ ratio.

APPLYING THE MODEL

Since it is necessary to know the entire sequence of letters to properly model a stroke, assessing the speed of stroking requires a word lexicon, rather than just a digraph table necessary to model tapping. Finding a suitable lexicon is complicated as different styles of writing use different vocabularies. Lexicons based on Twitter posts, Wikipedia articles, and the UIST Proceedings would all be expected to vary.

For this analysis, I use the 2006 list of the 40,000 most popular words in the English version of Project Gutenberg[2] (PG).[3]

[2] http://www.gutenberg.org

[3] Available at http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists

PG contains a digital library of over 24,000 full texts of books in the public domain. As these are typically older works, the lexicon overrepresents language from a previous era (e.g., "hath" is the 534th most popular word). While this is a disadvantage for modeling modern language, previous work [30] has demonstrated that even extremely diverse English-language lexicons have similar digraph patterns. The PG lexicon does have two significant advantages. First, it is an extremely large and diverse sample. Second, the lexicon includes a precise count of how much each word is used. Word frequency is necessary to determine how much impact the time to enter a particular word has on the result.

One complex part of using a lexicon is that not all the characters inside a word are letters. The PG lexicon contains hyphens ("old-fashioned"), abbreviations ("etc."), contractions ("don't"), and possessives ("man's"). While these only affect a small percentage of words (0.54% of the impact), they must still be modeled. For stroking, I have chosen to ignore these special characters as the mapping algorithm between a stroke and a word can figure out when these characters are necessary. For tapping, I have included the symbols as themselves if that symbol explicitly exists in the layout or as a space key otherwise. Another consideration for processing a lexicon is dealing with non-alphabetic letters ("naïve"). For stroking, the closest alphabetic neighbor (i for ï) can be stroked and then the mapping algorithm can choose the correct variant. For tapping, the solution is not so clear. One possibility is to model the taps required to achieve that character by transitioning into an optional keyboard. For this analysis, the point is moot since the PG lexicon does not contain any special characters, choosing instead to use the anglicized spelling of the word ("naive"). In addition, capitalization is important. For stroking, commonly capitalized words ("Europe" or "English") will automatically be capitalized by the mapping algorithm. For tapping, the user would have to hit the shift key first. To simplify the analysis, capitalization has been ignored. Finally, besides words, sentences contain punctuation. For this analysis, words are assumed to only be separated by spaces (the most common option), rather than commas, periods, paragraph breaks, etc. For stroking, spaces are automatically inserted between words. For tapping, the spaces must be tapped. This simplification is a common practice for modeling keyboard performance.

Evaluating Existing Layouts

To evaluate the keyboard layouts in Figure 1 for tapping and stroking, a few adjustments had to be made. First, keyboards with hexagon-shaped keys were modeled so that each key would occupy the same area as the square-shaped keys. That choice slightly favors the square-shaped keys. An alternative favoring the hexagon-shaped keys would be to use minimum key width (i.e., the distance between adjoining keys); this would be a legitimate alternative as tapping patterns tend to concentrate at the center of a key and drop significantly as they reach further away [35]. Equivalent area was also used for the unusually shaped Cirrin keyboard. For tapping, handling spaces is an important consideration. Keyboard layouts that feature multiple spaces (e.g., Fitaly) were modeled to choose the nearest space key after the last letter. This is a generous choice considering that actual users are not that adept at hitting the optimal space key [17]. The same strategy was used for the duplicated letters in the Montgomery layout.

Table 3 shows the predictions the model makes for the keyboard layouts. The measurements are given in words per minute (wpm), which, as usual for typing speed is actually characters (i.e., including spaces) per minute divided by five. As a first measure of how suitable a keyboard is to single-finger entry, I calculated the average distance between letters in a word. The different layouts vary widely with Dvorak the worst at 425% and OPTI II the best at 175%. From there, I calculated the tapping speed. Here the range was much smaller with Hexagon Qwerty the worst at 32.4 wpm and GAG II the best at 43.0 wpm. As Quikwriting maps a letter to a movement rather than a specific location, tapping cannot be modeled. These tapping results follow the previously published results. GAG I improves on Hooke and GAG II improves on Metropolis [24]. ATOMIK, with its alphabetical adjustment, is predicted to be slower than Metropolis II [34]. As the stroking model was based on right-handed user, I calculated the expected typing speed for left-handed users and right-handed users separately. Once again, Dvorak was the worst at 36.2 wpm and OPTI II was the best at 52.6 wpm. Finally, I calculated the ratio of the stroking time divided by the tapping time. For each layout, stroking was predicted to be faster. While tapping should be faster for longer distances, the crossover point may not have been reached for any of the layouts. In addition, the stroke technique benefits from not having to enter a space between words. The smallest gain was made by Dvorak, where stroking was only 9.7% faster; the largest gain was made by Square ATOMIK, where stroking was 40.8% faster. As $b_S > b_T$, keyboards with a smaller average key distances benefit more from stroking and there is a noticeable inverse correlation between these two percentages.

The wide-format Qwerty, Dvorak, and Wide Alphabetic performed poorly in every category, being in the bottom four each time. Dvorak performed particularly badly, testing the worst in all but one category. This is to be expected as Dvorak was designed so that a ten-finger typist would tend to alternate hands when typing letters. For stroke-based text entry, this feature results in a large number of long horizontal segments. This result is a great example of how a layout optimized for one purpose (ten-finger tapping) might be seriously problematic for a different input technique (one-finger stroking). Of the wide layouts, only the Chubon layout performed well. The other keyboard in the bottom four was Hexagon Qwerty, so just making the layout more compact is not sufficient. In contrast, Quasi-Qwerty fared fairly well. While the Montgomery layout improved substantially from tapping to stroking, it was still behind other layouts. Neither Cirrin nor Quikwriting performed well.

Most compact layouts tested well. For Square ATOMIK, the movement of the space bar from the center to the right side demonstrates the importance of the space key to tapping. The original ATOMIK layout with a space in the middle easily beats the square variant for tapping, whereas the square variant easily beats the original for stroking. This also explains

Keyboard Layout	*Key Distance / #Letters - 1*	*Tapping Speed*	*Stroking Speed (left-handed)*	*Stroking Speed (right-handed)*	*Stroking Speed / Tapping Speed*
Qwerty	320%	34.7 wpm	40.6 wpm	40.7 wpm	117.3%
Dvorak	425%	33.0 wpm	36.3 wpm	36.2 wpm	109.7%
Wide Alphabetic	344%	34.3 wpm	39.8 wpm	39.5 wpm	115.4%
Montgomery	238%	35.6 wpm	46.0 wpm	46.0 wpm	129.4%
Getschow et al.	187%	37.8 wpm	51.0 wpm	50.8 wpm	134.5%
Chubon	195%	39.2 wpm	50.4 wpm	50.2 wpm	127.9%
Fitaly	192%	41.9 wpm	50.4 wpm	50.4 wpm	120.3%
Cirrin	284%	35.4 wpm	43.0 wpm	43.0 wpm	121.6%
Quikwriting	310%	n/a	28.3 wpm	28.3 wpm	n/a
OPTI I	185%	41.9 wpm	51.3 wpm	51.3 wpm	122.5%
OPTI II	175%	41.6 wpm	52.6 wpm	52.7 wpm	126.5%
Lewis et al.	197%	37.6 wpm	50.0 wpm	50.0 wpm	133.1%
Square Alphabetic	259%	35.8 wpm	44.1 wpm	44.0 wpm	122.9%
Metropolis I	229%	38.6 wpm	46.9 wpm	46.5 wpm	120.6%
Hooke	236%	38.1 wpm	45.9 wpm	46.2 wpm	121.3%
Metropolis II	221%	39.2 wpm	47.4 wpm	47.5 wpm	121.1%
ATOMIK	234%	38.5 wpm	45.8 wpm	46.2 wpm	119.8%
GAG I	204%	40.1 wpm	49.0 wpm	49.2 wpm	122.6%
GAG II	187%	43.0 wpm	51.5 wpm	51.3 wpm	119.2%
Square ATOMIK	190%	36.0 wpm	50.5 wpm	50.6 wpm	140.8%
Hexagon Qwerty	334%	32.4 wpm	39.3 wpm	39.4 wpm	121.5%
Quasi-Qwerty	249%	37.0 wpm	45.6 wpm	45.6 wpm	123.3%
Square OSK	173%	40.9 wpm	52.8 wpm	53.0 wpm	129.6%
Hexagon OSK	174%	37.6 wpm	52.3 wpm	52.6 wpm	139.2%

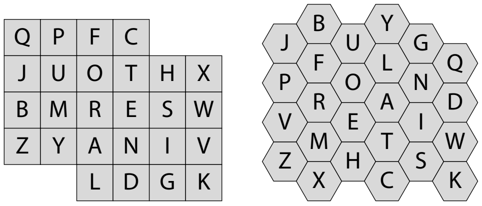
Table 3: Comparison of Different Keyboard Layouts

the large 40.8% gain that the Square ATOMIK made when transitioning from tapping to stroking. The best layout for tapping (GAG II) was the second best for stroking. The best layout for stroking (OPTI II) was the fourth best for taping. Overall, the keyboards that do well for one one-finger technique do well for the other.

Generating a Better Layout

The stroking model presented here can be used to evaluate an arbitrary keyboard layout. While it would take a human about 20 hours to stroke all the words in the PG lexicon, a modestly fast personal computer can model stroking the entire lexicon in less than 8 seconds. Theoretically, the model can determine the optimum arrangement of the 26 letters for a specific layout. Unfortunately, an exhaustive search would require 26! evaluations; at 8 seconds per evaluation, this would require 10^{20} years.

While an exhaustive search is prohibited, a simple optimization algorithm can still lead to acceptable results. While 8 seconds does not seem that much time, it takes much longer (more than 10,000 times as long) than evaluating a simple Fitts's law equation to model tap-based text entry; therefore, an intensive optimization technique is impractical. For instance, the genetic algorithm used to successfully generate GAG I and GAG II used 20,000 instances in the first generation [24]; at 8 seconds per evaluation, the first step alone would take almost two days to compute. Instead, I employed a simulated annealing process, followed by hill climbing. This algorithm was run several times for two pre-determined layouts: 1) a square-shaped key pattern using the same layout as Square ATOMIK; 2) a hexagon-shaped key pattern with a similar shape. After running both procedures multiple times, both of the optimized layouts (Figure 5) beat the most-efficient preexisting layout for those key shapes: Square OSK is 0.5% faster than OPTI II; Hexagon OSK is 7.0% faster than GAG I.



(a) Square OSK (b) Hexagon OSK

Figure 5: Generated Optimized Stroking Keyboards

DISCUSSION

The model of stroke timing introduced in this paper provides a useful tool for comparing the performance of different keyboard layouts for stroke-based text entry; however, any model has its limitations. For instance, there were significant performance differences between study participants; therefore, faster users might exceed the limits predicted by a model based on the average. In addition, model-based evaluations are a function of the inputs into that model (i.e., constants arrived from user data, digraph table / lexicon, how to model different key shapes). These inputs depend on context. The expected lexicon changes with task and author.

Contrary to published results, the OPTI II layout outperformed the Metropolis layouts for tapping in this analysis. The model introduced in this paper was created for a touch-based tabletop. The Metropolis layouts were based on Fitts's law constants for handheld devices. That the constants are different for the two devices is expected as stroking on a smaller surface involves a different muscle group (i.e., much more small finger movements, less large arm movements). As evidenced by the fact that most of the keyboard layouts evaluated in this paper were designed for these smaller devices, the interest in text-entry solutions for these devices is high. To serve that need, a similar study is being planned using iPods touch.

While expert speed is an important criteria, several other criteria are necessary to allow stroke-based text entry to be successful. While these are beyond the scope of this paper, their importance is noted here. First, word-level stroke-based techniques contain ambiguity. For instance, the stroke sequences for "Ted" and "tend" are identical for Square OSK; the stroke sequences for "to" and "too" are identical for any layout. A stroke-based keyboard must have a useful algorithm for mapping the stroke sequence to a word [31, 13] and must provide a usable interface for resolving ambiguity. While this might slow down stroking, other features (e.g., assisting with spelling, automatic capitalization, a mapping algorithm that does not require that the keys are hit exactly) will increase the expected speed of stroking. While this analysis predicts that stroking will be faster than tapping, this is based on tapping with one finger. It is fairly common for handheld users to use two fingers (or thumbs) to tap. A different model would be needed to assess the performance of a keyboard layout for tapping with multiple fingers. Another important category for any new method of text-entry, whether a new layout or technique, is how users learn the technique and how the technique deals with user errors [35].

CONCLUSION

While it is possible to design a more effective keyboard layout, getting others to adopt it is difficult. While Dvorak has been shown to consistently outperform Qwerty, Dvorak use is minimal. Why is that? First, switching to a new layout is time consuming and cognitively taxing [26]. Second, Qwerty is not a terrible layout. Given that typical improvement claims for Dvorak are only around 10% [26, 28], speed gain is not a great incentive to switch. For users hoping to address RSI problems, the increase in comfort and a decrease in fatigue associated with switching to Dvorak may be more important. Third, the default hardware uses a Qwerty layout. For typewriters, this was a particularly high hurdle since a change in layout usually necessitated buying a new typewriter. Computers have long had the ability to remap the key mappings with software, but this still leaves the marking of the keyboard the same. Because computer keyboards come with Qwerty labelling, many users probably do not realize that there is a better option.

For new technologies, such as handheld devices and interactive tabletops, there is an opportunity to introduce a new layout and a new text-entry technique as long as it improves performance. While novice users strongly prefer a Qwerty layout and are significantly faster with it initially [15, 18], it takes only about four hours of time for a more optimum layout to catch up [17]. The break-even point takes much longer when switching from Qwerty to Dvorak for ten-finger typing [26]. The model introduced in this paper predicts that a switch from tapping keys on Qwerty to stroking them will bring a speed improvement of 17.%. While that is a large improvement as far as keyboard performance goes, a substantially larger gain can be achieved by simultaneously switching to a different layout. Stroking with OPTI II is predicted to be over 50% faster than tapping with Qwerty. That may be a big enough incentive to entice users to switch layouts and techniques.

ACKNOWLEDGEMENTS

This work was done as part of the ShareIT project, a collaboration between the Open University's Pervasive Interaction Lab and the University of Sussex's Children and Technology Lab, funded by the EPSRC (grant number EP/F017324/1). I thank members of that project for their feedback on this research. I thank the reviewers of this paper for assisting me in better situating the work. I thank Circle Twelve for loaning us a DiamondTouch tabletop.

REFERENCES

1. J. Accot and S. Zhai. Beyond Fitts' law: Models for trajectory-based HCI tasks. In *Proceedings of CHI '97*, pages 295–302, New York, 1997. ACM Press.

2. J. Accot and S. Zhai. More than dotting the i's — foundations for crossing-based interfaces. In *Proceedings of CHI '02*, pages 73–80, New York, 2002. ACM Press.

3. X. Bi, B. A. Smith, and S. Zhai. Quasi-qwerty soft keyboard optimization. In *Proceedings of CHI '10*, pages 283–286, New York, 2010. ACM Press.

4. X. Cao and S. Zhai. Modeling human performance of pen stroke gestures. In *Proceedings of CHI '07*, pages 1495–1504, New York, 2007. ACM Press.

5. S. J. Castellucci and I. S. MacKenzie. Graffiti vs. unistrokes: An empirical comparison. In *Proceedings of CHI '08*, pages 305–308, New York, 2008. ACM Press.

6. R. A. Chubon and M. R. Hester. An enhanced standard computer keyboard system for single-finger and typing-stick typing. *Journal of Rehabilitation Research and Development*, 25(4):17–24, 1988.

7. P. Dietz and D. Leigh. DiamondTouch: A multi-user touch technology. In *Proceedings of UIST '01*, pages 219–226, New York, 2001. ACM Press.

8. C. O. Getschow, M. J. Rosen, and C. Goodenough-Trepagnier. A systematic approach to design a minimum distance alphabetical keyboard. In *Proceedings of RESNA '86*, pages 396–398, 1986.

9. U. Hinrichs, M. Hancock, S. Carpendale, and C. Collins. Examination of text-entry methods for

tabletop displays. In *Proceedings of TABLETOP '07*, pages 105–112, Los Alamitos, CA, 2007. IEEE.

10. U. Hinrichs, H. Schmidt, T. Isenberg, M. Hancock, and S. Carpendale. BubbleType: Enabling text entry within a walk-up tabletop installation. Research Report 2008-893-06, Department of Computer Science, University of Calgary, Alberta, Canada, February 2008.

11. M. Hunter, S. Zhai, and B. A. Smith. Physics-based graphical keyboard design. In *Proceedings of CHI '00*, pages 157–158, New York, 2000. ACM Press.

12. M. Kölsch and M. Turk. Keyboards without keyboards: A survey of virtual keyboards. Technical Report 2002-21, UCSB, Santa Barbara, CA, July 2002.

13. P.-O. Kristensson and S. Zhai. SHARK^2: A large vocabulary shorthand writing system for pen-based computers. In *Proceedings of UIST '04*, pages 43–52, New York, 2004. ACM Press.

14. J. R. Lewis, P. J. Kennedy, and M. J. LaLomia. Development of a digram-based typing key layout for single-finger/stylus input. In *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*, pages 415–419, 1999.

15. J. R. Lewis, M. J. LaLomia, and P. J. Kennedy. Evaluation of typing key layouts for stylus input. In *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*, pages 420–424, 1999.

16. I. S. MacKenzie. Fitt's law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7:91–139, 1992.

17. I. S. MacKenzie and S. X. Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of CHI '99*, pages 25–31, New York, 1999. ACM Press.

18. I. S. MacKenzie, S. X. Zhang, and R. W. Soukoreff. Text entry using soft keyboards. *Behaviour & Information Technology*, 18(4):235–244, 1999.

19. J. Mankoff and G. D. Abowd. Cirrin: A word-level unistroke keyboard for pen input. In *Proceedings of UIST '98*, pages 213–214, New York, 1998. ACM Press.

20. E. B. Montgomery. Bringing manual input into the 20th century: New keyboard concepts. *Computer*, 15:11–18, 1982.

21. D. A. Norman and D. Fisher. Why alphabetic keyboards are not easy to use: Keyboard layout doesn't much matter. *Human Factors*, 24(5):509–519, 1982.

22. R. Pastel. Measuring the difficulty of steering through corners. In *Proceedings of CHI '06*, pages 1087–1096, New York, 2006. ACM Press.

23. K. Perlin. Quikwriting: Continuous stylus-based text entry. In *Proceedings of UIST '98*, pages 215–216, New York, 1998. ACM Press.

24. M. Raynal and N. Vigouroux. Genetic algorithm to generate optimized soft keyboard. In *Extended Abstracts of CHI '05*, pages 1729–1732, New York, 2005. ACM Press.

25. K. Ryall, M. R. Morris, K. Everitt, C. Forlines, and C. Shen. Experiences with and observations of direct-touch tabletops. In *Proceedings of TABLETOP '06*, pages 89–96, Washington, DC, 2006. IEEE Computer Society.

26. E. Tenner. *Our Own Devices: The Past and Future of Body Technology*. Alfred A. Knopf, New York, 2003.

27. Textware Solutions. The Fitaly one-finger keyboard. `http://www.textware.com/fitaly/fitaly.htm`, 1998.

28. H. Yamada. A historical study of typewriters and typing methods: From the position of planning Japanese parallels. *Journal of Information Processing*, 2(4):175–202, 1980.

29. S. Zhai, M. Hunter, and B. A. Smith. The Metropolis keyboard—an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of UIST '00*, pages 119–128, New York, 2000. ACM Press.

30. S. Zhai, M. Hunter, and B. A. Smith. Performance optimization of virtual keyboards. *Human-Computer Interaction*, 17:229–269, 2002.

31. S. Zhai and P.-O. Kristensson. Shorthand writing on stylus keyboard. In *Proceedings of CHI '03*, pages 97–104, New York, 2003. ACM Press.

32. S. Zhai and P.-O. Kristensson. Introduction to shape writing. In I. S. MacKenzie and K. Tanaka-Ishii, editors, *Text Entry Systems: Mobility, Accessibility, Universality*, pages 139–158. Morgan Kaufmann, 2007.

33. S. Zhai and P.-O. Kristensson. Interlaced QWERTY — accommodating ease of visual search and input flexibility in shape writing. In *Proceedings of CHI '08*, pages 593–596, New York, 2008. ACM Press.

34. S. Zhai and B. A. Smith. Alphabetically biased virtual keyboards are easier to use: Layout does matter. In *Extended Abstracts of CHI '01*, pages 321–322, New York, 2001. ACM Press.

35. S. Zhai, A. Sue, and J. Accot. Movement model, hits distribution and learning in virtual keyboarding. In *Proceedings of CHI '02*, pages 17–24, New York, 2002. ACM Press.

36. S. X. Zhang. A high-performance soft keyboard for mobile systems. Master's thesis, Guelph University, 1998.