19

Text Input

A fundamental component of human thought and communication is language. Language provides the ability to pass knowledge and experience between people in very powerful ways. Yes “a picture is worth a thousand words” but it takes clear and insightful prose to give that picture meaning in ways beyond that snapshot in time. The whole history of scientific and social progress is founded on the power of language to communicate feelings and ideas across time and space. Language is a way for people to communicate connections and associations (sentences and paragraphs) among concepts and ideas (words and phrases).

Because of the huge importance of language our computers need the ability to enter text. This involves not only the creation of novels, texts and letters but more personal communications such as email or text messaging. Writing systems fall into two broad categories, ideographic and phonetic. In the ideographic systems, such as Chinese characters or Egyptian hieroglyphics, each character represents an entire word, syllable or idea. Chinese characters are not monolithic pictures. They are generally spatial rather than syntactic compositions of smaller symbols. Ideographic systems are generally more compact in that they communicate more ideas per square inch of display space.

Many modern writing systems are phonetic. In these systems there are a small number of characters (26 for English) that roughly correspond to sounds. By stringing the characters together we get a symbolic description of what the word should sound like. However, this phonetic description is not a pure one as any English speaker knows. Languages like Spanish are very phonetic where reading leads almost directly to pronunciation. The accent may be wrong but the pronunciation can be accurately reproduced from the written word. French is less so because in many cases some of the letters are not pronounced at all. English is downright chaotic because it is a hybrid of Celtic, Saxon, Danish, Latin, and French each contributing their own spelling patterns.

Phonetic writing systems have the property of being able to compose language from a relatively small number of symbols (less than 100). This allows practical keyboard designs. A keyboard for 10,000 symbols is impossible. Most of this chapter will focus on phonetic writing systems with ideographic approaches being addressed at the end.

Text input systems depend very much upon the physical situation in which the user is attempting to create text. If one is sitting at a desk with lots of room the keyboard is by far the most rapid way to create the written word. It is far faster for trained people to type than it is to write by hand. Most people do not always live, work or play in front of a desk or in a seated position. In particular, mobile computing creates situations that are rarely appropriate for keyboard use. There are two major approaches for text entry without a keyboard: use of a small number of buttons on a small device and use of a pen or stylus on the surface of a tablet. Another issue with mobile text entry is the ability to enter text without looking at the keyboard. Any typing or piano teacher will take great pains to train their students to look at the page, not the input device.

Before launching into a discussion of various input techniques we must first look at the nature of language. Text input is not random combinations of characters. Language has structure. Many text input techniques exploit this structure and we need algorithm models for representing language structure appropriately. Secondly we need metrics for evaluating the quality of text input techniques so that we can compare them. Following these two foundation sections we will discuss keyboard, button and pen techniques each in turn. Many pen techniques depend upon recognition of pen gestures. We will defer the recognition discussion to chapter 20 where we deal with digital ink in general. Lastly we will discuss the special needs of ideographic writing systems.

Nature of Language

For someone educated in a European language it is generally easy to classify a document as English, French, German, Italian or Spanish without knowing how to read or speak the language. They all use pretty much the same character set but the structure of words and the composition of words into sentences is very different among these five. A people we assemble the letters into sounds (probably incorrectly) and find that they “do not fit our tongue.” We hear people speak and say “that sounds Italian” though we cannot understand a word of it. All of this is because language has structure.

Language structure is very important when people recognize speech because it allows us to disambiguate what we hear based on what can reasonably go together. This structure is what allows us to read bad handwriting. We discard interpretations of letters that make no sense in our language (such as “xrtq” in English) and look for alternative interpretations. We consider alternative words based on how well the “fit” in a sentence. The structure of language makes communication more resilient in the presence of small errors. There are two main approaches to representing structure in language: syntax and probability.

Syntax

The study of syntax has a very old history in computer science due to its essential role in programming languages. Syntax is based on the idea that there is a *grammar* that defines the set of acceptable inputs in a language. Historically grammars are defined by automata. Finite-state and context-free are the most common representations. Finite-state and context-free automata have very efficient algorithms for testing an input sequence to see if it is acceptable by a grammar and for extracting structure from the input sequence.

An important property of an interactive syntactic system is that be *deterministic*. That is for a given state of the input and a given input from the user there is exactly one response/next state. This is important interactively for two reasons: 1) users are uncomfortable when a computer responds ambiguously and 2) trying all possible combinations is frequently impossible in interactive time. The second issue is fading in importance as computers get faster, but on PDAs and cell-phones with limited computing power, it is still an issue.

Any finite-state model can be automatically converted to a deterministic model that recognizes an equivalent[1] set of inputs. This means that every input can be handled in constant time, which is very important for interaction. There is no such algorithm for context-free grammars. Therefore there is no constant time processing for any context-free representation. Most use of syntax for text input uses finite-state models.

A most common use of finite-state representations is for encoding a dictionary of possible words. If we know the set of all possible words, then we can use it as a representation of appropriate language structure. A simple algorithm for searching a dictionary for a word W is to compare each word in the dictionary against word W. If there are N words in the dictionary then the cost of this check is N*length(W). With a large dictionary that is a problem. However, if we model our dictionary as a grammar for which each word is an acceptable sentence. The dictionary becomes a non-deterministic finite state grammar. The non-determinism comes from the fact that given the first input, many words are possible. Because the grammar is finite state we can always convert it to a deterministic representation where each input character can be handled in constant time. This makes the dictionary lookup problem of order length(W) regardless of how many items there are in the dictionary. See the appendix A3.2b for an example.

The notion of a grammar for text input is very problematic for interactive text input. People do not spell correctly (violating the dictionary). People do not use correct grammar either accidentally, through ignorance or on purpose. People continuously invent new words, phrases and abbreviations. This makes grammars with their precise definitions of what is and is not acceptable a rather brittle representation.

Probabilistic Language Models

An alternative way to capture the structure of language is through statistics. For example the space character and the vowels account for 49.9% of all English character usage[2]. Obviously an input technique that made spaces and vowels easier to enter would do better than a similar technique without such an adjustment. The structure of language is modeled more deeply in the sequences of letters and words rather than individually.

The most common statistical device for modeling language are the *N-grams*. An N-gram is a sequence of N tokens as found in some corpus of text. Sometimes the N-grams are sequences of letters and sometimes sequences of words. Though the data are different, the techniques of analysis are the same. For example, to compute the digram (2-gram) probability of ASCII characters, we can construct a 128 x 128 table *D[i,j]* and use it to count the number of character combinations in a large corpus of text. We can use this table of counts to compute various probabilities. In many cases this 128x128 table is restricted to just letters without differentiation of case.

Suppose we are given a digram frequency table *D* and the user has previously entered the character ‘x’. We may then want to compute the probability that ‘o’ is the next character to be entered using the formula

$$P('o'|'x') = \frac{D['x','o']}{\sum_{i=0}^{127} D['x',i]}$$

Soukoreff and MacKenzie have published the digram frequencies for English letters[2]. These frequencies change, however, depending on the purpose for text. The character frequencies that occur in instant messaging are different from those in more traditional prose.

Digrams do not provide a lot of structural information. We can get more structural information using higher values for *N*. The size of the N-gram table grows as L^N where *L* is the number of possible tokens and *N* is the length of each N-gram. For characters a trigram (3-gram) table easily fits in RAM. For the 10,000 most common words, the trigram table would be 10^{12} or well over a terabyte. A table this size also poses a statistical problem. To get a reasonable sample we would like 10-20 samples per table entry. At 5 characters per word (in English) this means a corpus of nearly a petabyte.

Fortunately most linguistic data follows Zipf's power law. This says that when items for sorted from most frequent to least frequent they follow a power curve r^{-b} where *r* is the rank of the item and *b* is a positive constant near 1.0. Figure 1 shows such a power curve. What this tells us is that a few things happen at a very high frequency and this tails off into many very infrequent items. This is consistent with the notion that language has a great deal of structure. There are not actually 10^{12} trigrams in English. The combination "seven, pig, fluorine" just never occurs in a sentence.

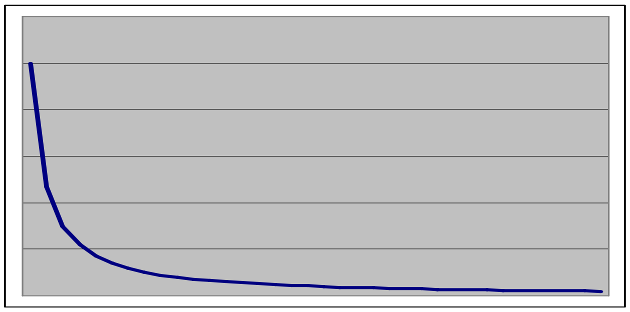


Figure 1 – Power curve of item frequency

This structure in the N-gram representation is dealt with in two ways. The first is that for infrequent words a lower order (N-1)-gram or (N-2)-gram is used. This lets us use the structure information of an N-gram without requiring a huge corpus[3]. For the frequent combinations like "in the end" we can use trigrams and for less common phrases like "lateral air stabilization" we can use digrams or simple word frequencies. Secondly we can use highly sparse representations of our N-gram table so that it actually fits into memory and is usable.

Using token frequencies we can make predictions about a user's next inputs and optimize our input techniques for actual user behavior. An early example of this was Witten's Reactive Keyboard[4] that used letter trigrams to speed the typing of programs where common words like variable and function names occur with high frequency. In this case the corpus was the program files on which the user was currently working.

Evaluating text input

As we look at various ways for entering text we need to have metrics that we can use to compare their performance. The two obvious measures are speed and errors. We want techniques that are as fast as possible with as few errors as possible. The standard for text entry speed is *words per minute*. This is measured as characters per minute divided by the average word length (5 characters in English).

Measuring speed

Measuring speed is problematic because of training effects. There is a huge difference between a beginning typist and a trained touch typist. When proposing a new input technique we would like to know the speed that one can reach after training and the amount of practice that it will take to reach that level of proficiency. We can measure expert input speed by having subjects regularly practice the technique until their speed stops increasing. We can then assume that we have reached that subject's maximum speed. Experience with teaching school children to type shows that such speeds are reached after months of practice. We can measure learning as the amount of practice time required to reach this level.

It is very difficult and very expensive to get a statistically useful number of subjects to practice a new input technique for long enough to get expert speed numbers. Not only is the time difficult but many users are not motivated to reach maximum speeds as evidenced by the number people who persist in sub-optimal typing techniques. They know that there is a better technique but what they do now is good enough for their purpose.

To overcome these problems with experimental speed measures various proposals for theoretical measures of how fast one might be able to enter text. These theoretical measures of possible entry speeds are based on models of user behavior and user motor capabilities. When comparing the theoretical speeds of two input techniques we can assume that the user's motor capacities are the same between the techniques and the only differences are the techniques themselves. An early model for entry is the Keystroke Level Model (KLM)[5]. This model simply measures the number of keys that must be pressed to accomplish the goal. For example, on a normal keyboard the letter 'A' requires two keystrokes: the shift key and the 'a' key. The Silfverberg model[6] builds upon KLM by taking into account the positioning of keys and the time required to reach and select one of them. Because of varying distances the time to reach various keys is different, therefore Silfverberg takes the probability of various key combinations into account. James and Reischel[7] have published data on how these models compare to actual performance.

It should be noted that speed is not necessarily the primary consideration in a task involving text entry. For example, this chapter contains about 8,600 words and I type at about 50 words per minute. This means that actual typing time would be about 3 hours. However, it took about 20 hours to write this chapter. Text entry is about 15% of the total task. Changing my text entry to something twice as fast (not very likely) would still only improve my writing task by about 7%. That would not be a great motivation to retrain my typing. However, if I was text messaging this chapter using a phone keyboard at about 5 words per minute, the text entry time would be 30 hours rather than 3 and I would be very motivated to cut that time in half or better.

Some input techniques use ambiguous entries where each key corresponds to more than one letter. These techniques rely upon probability, dictionaries and language structure to resolve the ambiguity and eventually select the correct word for entry. One of the measures of such technique is *disambiguation accuracy*. This uses probability to measure the likelihood that the correct word will be chosen despite the character level ambiguity. A variant is the measure of the likelihood of the correct selection given *N* inputs. These techniques can frequently demonstrate fewer keystrokes than the number of characters in the word.

Measuring errors

Rapid entry of text is a good thing, but the actual goal is rapid entry of correct text. For this we need a measure of the errors that users make. The simplest error metric is to compare the string that the user entered with the desired string and count the number of differences. This measure has serious problems as shown in figure 2. With the simple character measure, all of the underlined characters are counted as errors when simple inspection shows that the only error was an erroneous ‘u’.

```
the lazy brown dog ate my lunch
the lazuy brown dog ate my lunch
```

Figure 2 – Character level errors

The problems with simple character comparison can be resolved by computing the Minimal String Distance (MSD) between the desired string and the actual string that was entered. MSD is a special case of the Minimal Edit Distance algorithm where all edit costs are 1. The Minimal Edit Distance algorithm is described in appendix A3.2d. The MSD is a count of the number of edits that must be made to convert one string into the other. In figure 2 the MSD is 1 (remove the 'u'). MSD is a much more accurate measure than simple comparison.

String comparisons like MSD do not account for the effort required to correct an error. For example the error in figure 2 could be corrected in many ways. If the error is recognized immediately then a rubout would remove it. If the error is not recognized until the end of the word then the correction might be "rubout, rubout, 'y'" or possibly "left arrow, rubout, right arrow". These kinds of behaviors are accounted for by the Key Strokes Per Character (KSPC) metric. A KSPC of 1.0 means that the entry was correct. Any errors and their corrections will cause a higher KSPC. Soukoreff and MacKenzie[8] propose an additional metric that unifies MSD and KSPC. This is expanded upon by Gong and Tarasewich[9].

Test data for measuring text input

With all of the metrics for speed, learning and errors the text itself matters. MacKenzie and Soukoreff[10] have published phrase sets that can be used as a standard for text entry. However, it has been shown[11] that the words people use when they write are different from spoken words and different still from the words people use when sending text messages on cell phones. It is very important to use a test set that reflects the kinds of phrases that are likely to be used. Drawing test phrases from the Wall Street Journal will not accurately reflect how teens send text messages to each other.

Keyboard input

The dominant tool for text input is the keyboard with each letter and number having its own key and a shift key for changing case. In English the most popular layout is the Qwerty keyboard that is so named because of the order of the first six letters on the top row. The design of the Qwerty keyboard was dictated by the mechanics of early typewriters and by a complete lack of understanding of modern ergonomics. With the digital age the mechanical limitations have long since disappeared.

More efficient key layouts have been proposed. The most common is the Dvorak keyboard. Dvorak designed his keyboard so that the most common characters would be on the home row and thus reduce finger movement. The layout is also designed so that successive keys will most probably occur on alternating hands. These innovations make input faster and produce less fatigue.

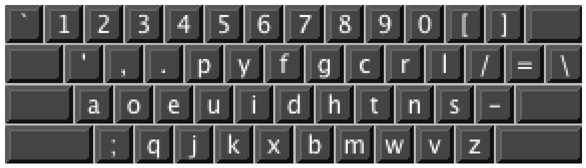


Figure 3 – Dvorak keyboard layout

Using standard Qwerty keyboards, typists can reach 60+ words per minute. As of 2005, the world record[12] is 150 words per minute using the Dvorak keyboard.

An alternative is the stenographer's keyboard used when close-captioning television programs or in court reporting. The stenographer's keyboard is phoneme (sound) rather than character oriented and it chorded. For a given syllable the recorder will press multiple keys simultaneously. For example the word "cat" would be entered by pressing "K", "A" and "T" simultaneously. This allows for much more rapid entry (225 words per minute with training) but does not produce exact text entry. Careful transcription and proofreading is required to produce the final text. Many stenographer's cannot read the input of another without effort because the inputs are phonetic and suggestive rather than exact representations of words.

Some keyboards have modified layouts that simplify typing with only one hand. This allows the other hand to work with a mouse or perform some other task. A novel approach to one-handed typing is the Half-Qwerty[13] keyboard. In this keyboard both the right and left hand portions of the Querty keyboard are mapped on to the same set of keys with a special shift key to move between the choices. It turns out that a surprising amount of muscle memory location of keys is transferred between the right and left hands.

Buttons

Many devices do not have the physical space required for a full keyboard and many other devices have teeny, tiny keys that are hard for adults to press. One solution to the size problem is to use fewer buttons than the size of the alphabet. There are a large number of methods that have been proposed for generating text input from a small number of controls. There are a collection of techniques that map multiple characters to the same key with various methods for disambiguating what the input should be and various methods for rearranging the character/key binding. Some have proposed chorded systems where the desired character is specified by pressing multiple keys at once. There are techniques for scrolling through character choices and lists that use the geometry of letters to define an input sequence.

Much of the effort is focused on the 12 key arrangement found on most telephones (Figure 4). The most common technique for text entry is Multitap where the user presses a key multiple times. For example, to enter the word “golf” the user would press 4,6,6,6,5,5,5,3,3,3. This has the advantage of being very easy to learn. It has the obvious disadvantage of requiring many more keystrokes per character (KSPC) than 1. There is also some remaining ambiguity in the input. For example the word “cat” would require 2,2,2,2,8. However, this sequence might also mean “bbt” or “act”. The problem is using the number 2 for successive letters. This is dealt with in two ways. The first is a timeout at the end of a character, for example 2,2,2,*pause*,2,8. This slows down entry but creates the boundary between “c” and “a”. The other alternative is to use a special key to move on to the next character such as 2,2,2,#,2,8. This has been shown to be much faster. Most Multitap system provide both. If one considers English character frequencies, MacKenzie et. al. [14] report that the average KSPC for Multitap is 2.0342.

1	2 abc	3 def
4 ghi	5 jkl	6 mno
7 pqrs	8 tuv	9 wxyz
*	0	#

Figure 4 – Telephone text key layout

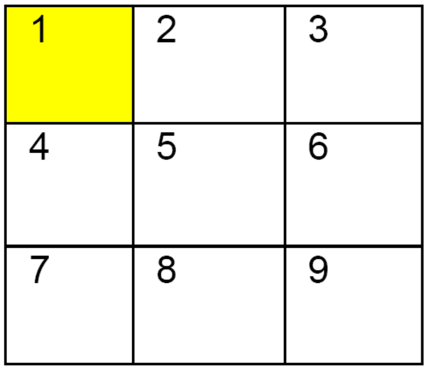
To resolve the KSPC of 2, the T9 technique was created by Tegic Communications. In this technique there is a dictionary of words. The keypad is the same as for Multitap but each key is only pressed once. For example "golf" would be 4,6,5,3. This sequence is highly ambiguous until one considers the structure of language. The string "hnke" is a possible interpretation of the input, but does not form an English word. By using the dictionary much of the ambiguity can be resolved. This is not a complete solution because "cat" and "bat" both use the sequence 2,2,8. The most probable word from the dictionary is shown and then the user can use another key to cycle through other alternatives. The most probable word prediction can be based on actual word probability or the use of N-grams to exploit words entered previously.

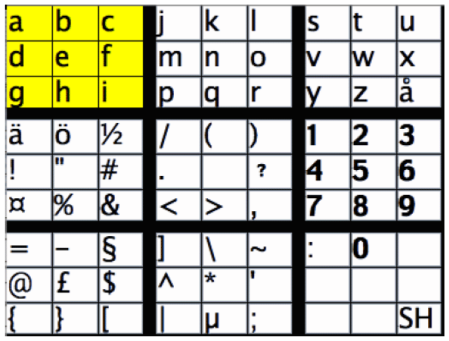
T9 has a KSPC of 1.0072 for English[14], which is very good. The problem is that the user is not sure what word has been entered until the entire word is complete and selected. It is very difficult to detect an entry error without the user doing mental disambiguation as they type. T9 also struggles with words, slang or abbreviations that are not in the dictionary. Most T9 systems allow for new words to be added (usually with Multitap) but the problem remains.

The LetterWise[14] system addressed these issues with T9 by using character N-grams rather than a dictionary. When the user presses a key the most probable character, given the previous N-1 characters, is displayed. This means that the user sees immediately the character that has been selected. If this character is incorrect, Multitap techniques are used to get at the correct character. This is an extension of the earlier LessTap[15] system that uses only character probabilities. LessTap and LetterWise are not reliant upon words being in the dictionary. MacKenzie et. al. have reported a KSPC for 1.15 for LetterWise. They also performed actual text entry experiments over many days to address the learning issues. In the first 25 minute session LetterWise achieved 7.3 words per minute with Multitap generating 7.2 wpm. By the 20th session LetterWise users achieved an average of 21 wpm with Multitap achieving 15.5 wpm.

Some researchers have explored the reallocation of the key/character binding. One option[16] is to retain the alphabetic ordering for ease of learning while moving characters to different buttons thus changing the number of characters assigned to each button. It is also possible to completely rearrange the character/button mapping. Experiments have shown that the complete remapping is harder for users to learn than the alphabetic form. Though these techniques produce much better disambiguation of words than the standard layout the KSPC is still only 1.16 which is not a significant advantage over LetterWise.

The previous techniques have all assumed small input devices with small screens. The TNT[17] approach assumes large screens with small controls as one finds with television remotes and game controllers. TNT uses an array of 9 keys to select up to 81 characters using 2 key combinations. The approach is based upon the displayed array shown in figure 5. The user's first key press will select one of the 9 regions and the second press selects one of 9 characters from that region. This technique has the same KSPC=2 as Multitap, but can enter many more characters than simple letters or numbers. This is particularly important in European languages with special character modifiers that push the character set beyond the basic 26 letters. Experiments show users reaching 9.52 wpm with some practice.





a	b	c	j	k	l	s	t	u
d	e	f	m	n	o	v	w	x
g	h	i	p	q	r	y	z	å
ä	ö	½	/	(	)	**1**	**2**	**3**
!	"	#	.		?	**4**	**5**	**6**
¤	%	&	<	>	,	**7**	**8**	**9**
=	–	§	]	\	~	:	**0**	
@	£	$	^	*	'			
{	}	[	|	µ	;			SH

Figure 5 – TNT Character layout.

Chorded text entry

One way to overcome the limited number of keys is to use a chorded input technique. Conrad and Longman[18] were among the first to experiment with this technique. A more recent attempt is the Twiddler[19], shown in figure 6. The arrangement is a row of three keys for each of four fingers so that the device can be operated with one hand. The keys are larger and farther apart than with phone keypads to facilitate one handed operation. An expert user can achieve 60 wpm, which is comparable to Qwerty keyboard entry. The problem is that the cording is difficult to learn. The Twiddler was compared experimentally with Multitap in a series of sessions. For each session the user typed for 20 minutes using one technique and the 20 minutes with the other. After 8 such sessions the Twiddler was faster on the average than Multitap and the error rates were below 5%.

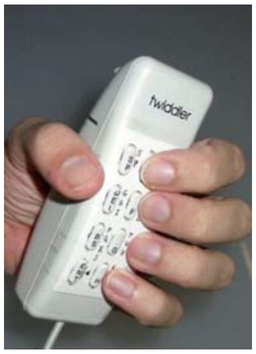


Figure 6 – Twiddler

The ChordTap[20] technique uses the phone keypad with the addition of three buttons on the back of the phone. The three buttons select which of the three characters on the phone pad should be selected. Any two of the chord buttons will select character 4. The additional buttons simply accelerate the use of the Multitap system. By the end of a series of trials, ChordTap achieved an average 16.06 wpm against Multitap's 11 wpm. The error rates for Multitap were 2.6% while those for ChordTap wer 3.3%. The big advantage of ChordTap over the Twidler is the ease with which users can learn how it functions. The direct mapping between the technique and the key labeling is very helpful.

Scrolling through choices

With a very limited number of buttons there are scroll selection techniques as are found in video games. The most common is the three-key technique where the user scrolls up or down through the alphabet using two of the keys and then uses the third key to select a letter. Users achieve 10 wpm after 10 training sessions[21]. A five-key technique uses four keys to scroll around a matrix of letters and one key to select. This technique achieved 13 wpm after 10 training sessions[21]. A further variation uses a game controller joystick to scroll through two different matrices, each with half of the characters. There are then the left and right selection buttons to indicate which matrix character to choose. This achieved 7 wpm[22].

Other inputs

There are a few techniques for text entry that use other inputs than keys. The TiltType[23] system uses four buttons and an accelerometer to select text. The accelerometer senses one of 8 tilt positions while the buttons select from 6 sets of 8 characters. A combination of tilting a button presses creates a text entry technique that fits into the form-factor of a digital watch. There are no speed measurements for this device and indications are that it is not very fast.

A second technique uses a touch wheel[24], which is a round surface that can sense where on the surface the user's finger is touching. This works somewhat like the scrolling techniques except that the characters are arranged around the wheel. The user can rapidly go near a character by starting at the appropriate location on the wheel and then moving the thumb until the correct character is selected. Using a probabilistic language model, characters that are more likely are easier to select than characters that are less likely, which increases speed. However, this simple language technique causes infrequent characters to be very difficult to select. The authors accommodate this by weighting the language model by the speed of the user's movement. The faster the user is scrolling the stronger the language model's preference. This corresponds to rapid normal input. As the user slows down the influence of the language model is diminished making rare characters easier to select. This corresponds to a more careful selection. This technique reports input speeds of 6.2 wpm.

Stylus

In many situations a keyboard is not practical such as many hand-held devices. One option that has received considerable attention is the use of a stylus or pen to input characters. There are four basic techniques in use. They are: soft keyboards, single character recognition, phrase recognition and cursive handwriting recognition. We will also look at a novel stylus-based scrolling technique.

Soft keyboards

A soft keyboard is simply a picture of a keyboard displayed on the screen where users can touch keys with the stylus. Experiments have shown[25] that users reach a speed of about 40 wpm on Qwerty-style soft keyboard layouts. One of the reasons that Qwerty keyboards have remained despite their shortcomings relative to other keyboards is the extensive typing training that people have invested in them. However, such training does not transfer directly to soft keyboards. The "muscle memory" effects that optimize typing do not apply to soft keyboards where very different muscle groups are used. In fact most touch typists cannot identify the location of a specific key on the Qwerty keyboard without using their hands to simulate typing that key or visually searching the keyboard.

Because key layout for soft keyboards does not have the heavy training associated with physical keyboards, researchers have explored more effective key arrangements. Using Fitt's law for time to select a target, MacKenzie and Zhang developed a metric for measuring the effectiveness of a layout[25]. By using a statistical language model for English they derived the probability of moving the stylus from one key to another. This probability was used to weight the Fitt's law prediction of the time to make that movement. A weighted average of such times was used to predict the speed of typing using a particular layout.

Using their metric they tried a wide variety of layouts and computed their predicted typing speed. They finally focused on their OPTI layout shown in figure 7. Note that because SPACE is such a common key, they included four separate space keys to make it much easier to enter and made each of them bigger so as to require less time to hit them accurately. In their experiments they showed that although users were initially slower with OPTI, they soon reached a speed of 45 wpm that is 10% faster than a Qwerty layout.

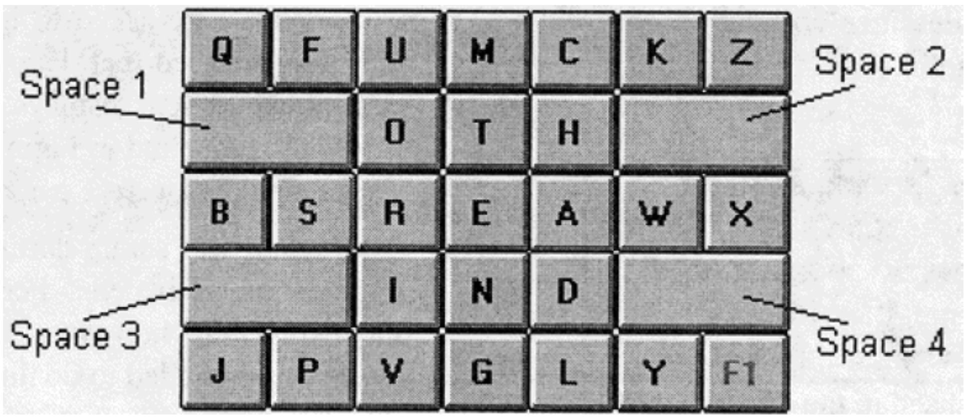


Figure 7 – OPTI soft keyboard layout[25]

Their approach for finding the best layout was to think of a design and then evaluate it with their metric and then think of another design. To take a more automated approach, Zhai et. al.[26] modeled their keyboard as a set of circles that could move freely in a 2D plane but could not overlap. They used a digram probability model for sequences of characters and a Metropolis optimization technique to move the keys around seeking a "minimal energy" configuration based on Fitt's law and the language model. They then translated the "minimal energy" layout using circular keys into a layout of hexagonal keys (hexagons are the most compact shape that completely fills a space) as shown in figure 8. Using a different prediction metric they reported speeds of 43 wpm on all of the best designs that their algorithms identified.

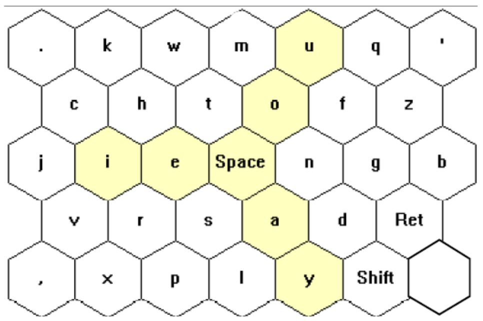


Figure 8 – Metropolis Keyboard[26]

The primary issue in alternative key layouts is to get the most common character pairs close together. Isokoski[27] devised a technique such that when a user selects a character, a menu of the most likely next characters pops up around the selection, as in figure 9. The next character can then be selected by a drag movement to the right choice or the stylus can be lifted and the popup menu ignored. One of the problems with adapting the popup to each letter is that the menu changes from letter to letter. Their solution was to put only the vowels and space key in the popup since they are the most frequent. This makes the popup stable and less confusing. Experiments showed that users took advantage of the popup shortcut 20% of the time and users were faster than simple Qwerty layouts after 20 sessions.

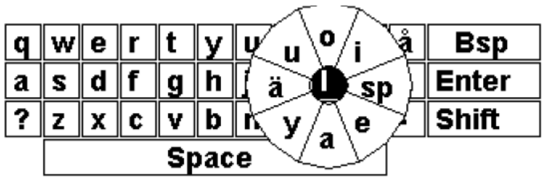


Figure 9 – Popup keyboard[27]

Masui pushed the notion of popups on soft keyboard's even further with his POBox[28] system. His approach was to use a language model based on word pairs rather than character pairs. Using the previous word and the characters already typed he would assemble a menu of the most probable words that the user was trying to type. The user could then select a word from this menu or continue typing. If there were not enough high probability words, he would add words that approximately match the characters being typed. A major focus of his work was rapid entry of ideographic characters using a small keyboard. With a menu of 10 words he is able to present the correct next word 53% of the time with only one character. His technique will present the correct word 90% of the time after only 3 characters. In his experiments he reported people entering text with pen and paper at a rate of 49 char/min. With POBox his users reached 40 char/min. This compares well with 20 char/min for systems using handwriting recognition.

Character input

Ever since it became possible to sense the location of a pen on a surface there has been the goal of users entering text by writing on the screen. This goal should be tempered by the fact that writing text with pen and paper is much slower and less legible than typing. This means that fundamentally, written character input is not the most efficient means even if it seems more "natural". However, there are many physical situations where a keyboard is too large to carry or too awkward to use. It is very difficult to use a keyboard while standing without a special tall desk. Two handed typing requires that the keyboard be supported by a desk or lap. In many of these situations, pen-input of text is more effective.

Character recognition is based on the simple algorithm, collect a digital ink stroke (all points between mouse down and mouse up), compute a set of features from the stroke, and match those features against known examples of various characters. The details of these recognition techniques will be discussed in more depth in Chapter 20 on Digital Ink.

Most of the work on pen entry of text has focused on the recognition of individual characters. This has the nice property of only recognizing a few tens of specific symbols for phonetic writing systems. Ideographic character recognition is much, much more difficult. One of the difficulties with character recognition is the segmentation of characters. If characters are written with 1-3 strokes of the pen, how does one identify when one character stops and the next stroke belongs to a new character? For example, capital "A" and a "t" are composed of an initial stroke and a second crossing stroke. The "t" could be mistaken for an "i" or an "l" followed by a dash "-". Various systems were proposed to resolve this including whether strokes overlapped in the X dimension or whether there was a pause between strokes. These solutions were not satisfactory because user's handwriting styles are never that tidy.

Goldberg and Richardson introduced the idea of not using the true handwritten character as the input but rather use a single stylized stroke that approximates the character. In their Unistrokes[29] system the strokes were selected to be easy for character recognizers to identify correctly. By using a single stroke per character the character segmentation problem is eliminated. The strokes were somewhat shaped like the characters that they represent, but not completely so.

One of the problems with Unistrokes was that users had a hard time remembering the correct stroke. Venolia and Neiberg addressed this problem by using techniques from marking menus (discussed in Chapter 21 on menu systems). Their T-Cube[30] system started with a simple pie diagram with 8 starting locations, as shown on the left of figure 10.

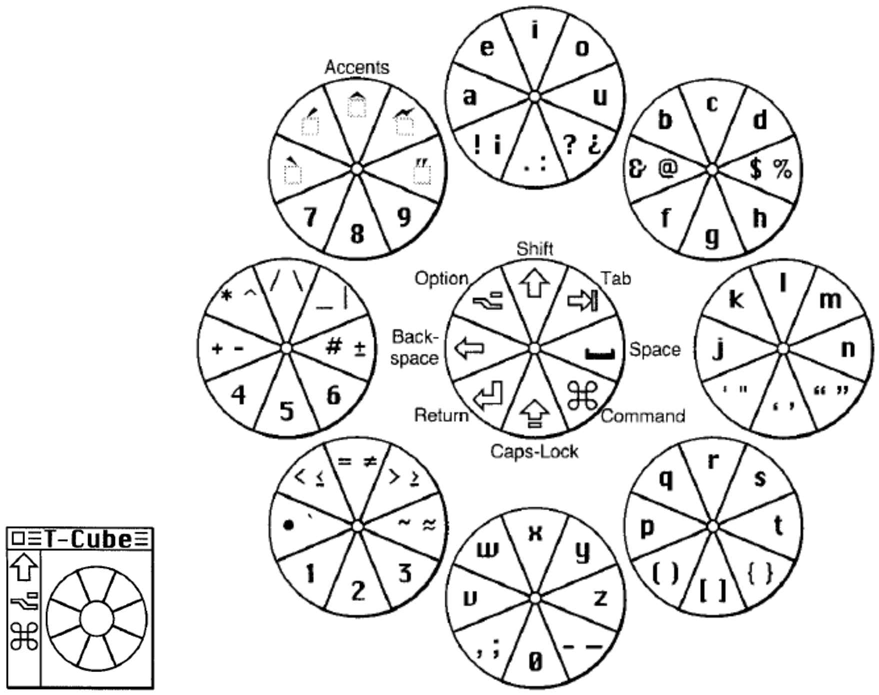


Figure 10 – T-Cube Menus[30]

A character is entered by placing the pen down in one of the nine regions of the basic shape. If the user pauses with the pen down, then one of the 9 menus shown in figure 10 will appear centered on the stylus and the user can select a character by moving in the correct direction before lifting the stylus. If the user does not pause but just makes the stylus movement, the character is selected without popping up the menu. The advantages are that characters are selected by single flicking stroke and that the system automatically trains the users as to the correct strokes using the popup menus.

Pen-based character input achieved commercial success with the Graffiti[31] system that came with Palm Pilot hand-held computers. This approach is an alphabet inspired by Unistrokes but with more memorable strokes that looked like the characters they represented. Graffiti also used ideas from T-Cube in that they used two regions separately for letters and numbers rather than a single region for all characters.

The EdgeWrite[32] system addresses the needs of users with impaired motor control. The authors were concerned about those with hand tremors or other problems with aging who could not enter Graffiti characters in an error-free manner. Their solution was to add a plastic template over the Palm's text entry region as shown in figure 11. The template provides an edge to guide the strokes. This means that rounded strokes are no longer possible. It also means that strokes are composed by moving from corner to corner rather than freely in the space.

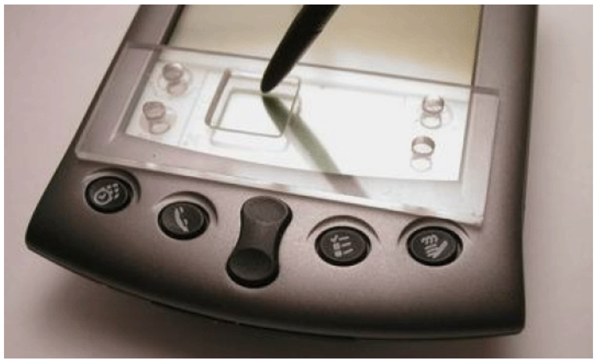


Figure 11 – EdgeWrite Template[32]

Because all movement is from corner to corner a character can be recognized by the sequence of corners that the stylus visits. This is a much simpler recognition algorithm. In testing EdgeWrite was comparable to Graffitti in terms of speed for unimpaired users and much better for motor impaired users. A portion of the EdgeWrite alphabet is shown in figure 12. The alphabet was formed by starting with Graffiti's unistroke stylizations of the characters and then adapting that stylization to corner sequences. The strokes sort of look like the characters they represent, which makes them easier to learn.

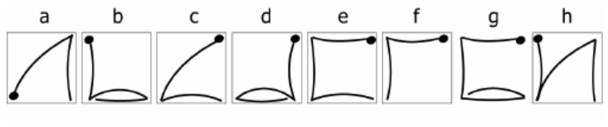


Figure 12 - EdgeWrite Alphabet[32]

Because the EdgeWrite alphabet is formed by sequences of corner visits, other sensing technologies besides the stylus are possible. Using four keys (one for each corner)[33] one can reproduce an EdgeWrite stroke by pressing the keys in the same sequence as the corners visited using a stylus. After 10 training sessions of 5 minutes each, users achieved speeds of 17 wpm using the four keys.

One can also view the EdgeWrite alphabet in terms of a sequence of directions rather than corners visited. Based on this observation the alphabet can be adapted to entry via a trackball[34]. Each direction is sensed as a "pulse stroke" that extends beyond a minimum distance. There are 8 possible pulse directions. If one considers the EdgeWrite alphabet and its corner orientation, there are only three possible directions from any given corner. This restriction makes the technique much more stable for those with motor impairments. Unimpaired users achieved speeds of 9.82 wpm and a user with a spinal cord injury achieved 6 wpm, which far exceeded his speed with other techniques.

Stylus word input

Several researchers have sought to increase stylus text speed by creating techniques for entering complete words with a single stroke. Two early techniques used a central "resting zone" surrounded by characters. Users begin their stroke in the resting zone and then "visit" the desired characters in the word in sequence. Figure 13 shows the stroke for the word "finished" using the Cirrin[35] system. Starting in the resting region each character is visited in turn. Note that characters that are adjacent both in the word and the key layout can be visited without returning to the resting region. The only data reported with Cirron is 20 wpm by a single experienced user. The claimed advantage is that a continuous stroke takes less time and effort then a series of tap and lift motions. Note that the letters are not arranged in alphabetical order. Instead they are arranged to minimize the probable distance traveled between characters based on a statistical language model.

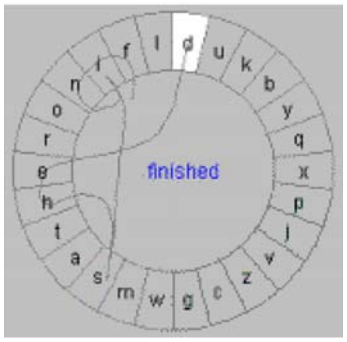


Figure 13 – Cirron stroke for entering the word "finished"[35]

Perlin's Quikwriting[36] takes a similar approach with a different layout and stroke style. His layout consists of 8 border zones, rather than one for each character, and a central resting zone. A stroke begins in the resting zone and then moves into one of the character zones. Each character zone is assigned an odd number of characters presented in an order around the periphery of the layout as shown in figure 14. If the stroke returns to the resting zone, then the central character in the zone is selected. If the stroke moves clockwise into another character zone rather than returning to the resting zone, then the next character in the clockwise direction is selected. Returning to the resting zone selects a character without lifting the stylus. This is best illustrated by the stroke for the word "the" as shown in figure 14.

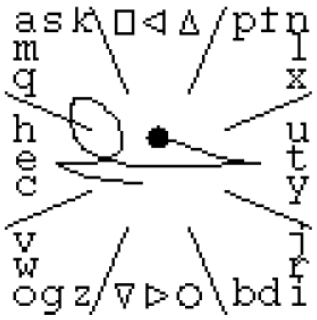


Figure 14 – The word "the" using Quikwriting[36]

No experimental data is reported for Quikwriting, but anecdotal evidence shows much higher speeds than Graffitti after significant practice. The claimed advantage is that users quickly learn the stroke for common words and do not consider individual characters any more. This allows for the kinds of training speedup that occurs with touch typists. This gesture speedup is less likely to occur with Cirrin because of the greater accuracy required to hit one of 26 zones rather than one of 8. Quikwriting would be harder to learn but more robust to fast but sloppy gestures.

The notion of a stroke gesture composed by movements among letter zones is carried even farther in Shumin Zhai's SHARK system[37]. In SHARK the idea is that moving among the letters of a word on a soft keyboard, without raising the stylus, creates a gesture with a characteristic shape for each word. In SHARK, these gestures are called *sokgraphs*. The SHARK approach is to compare a given gesture with example gestures from all of the words in a dictionary. By comparing their shapes rather than their positions, users can become increasingly sloppy about their accuracy while still entering words very rapidly. The idea is that a user begins text entry by tracing words on the keyboard. For common words, the user becomes faster and faster at tracing the stroke. Eventually the common sokgraphs become a shorthand for words that the user enters very rapidly.

Figure 15 shows SHARK's sokgraph for the word "system". The thick, straight line shows the exact sokgraph constructed from key center to key center. The thin, pen-like line shows how a user might rapidly draw the sokgraph. The shape that the user draws must be compared against the formal sokgraphs for all of the words in the dictionary. A dictionary of 10,000 is the minimum to be effective and the comparison can be very expensive. To be effective, SHARK must find the correct word in less than 1 second.

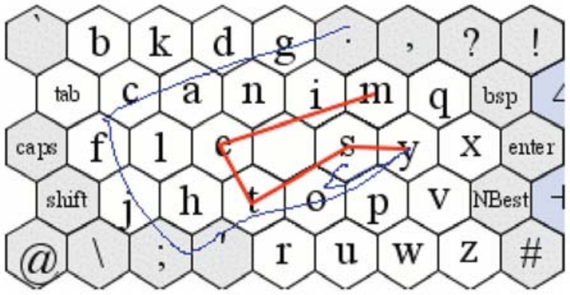


Figure 15 – SHARK Sokgraph for "system"[37]

The first step in SHARK's recognition is to normalize the strokes so that the largest X, Y dimension is 1.0 and the origin is normalized. This allows for stroke comparison without reference to size or position of the stroke. The positions of the start and end points of the strokes can be compared in normalized coordinates. This allows for very rapid pruning of the set of strokes to be compared. Candidate strokes are compared using “elastic matching” that compares closes points to closest points to come up with a distance measure. This measures the shape similarity of the two strokes. The strokes are also compared on the distances between their start and end keys. Even with this effort there are still words that share the same sokgraph. In Zhai’s experiments the Qwerty keyboard had 537 words with similar gestures. These problems are further reduced using a language model to predict the correct choice. Where confusion among words still remains a menu of choices is presented for selection. Experiments show “record” speeds of 70 wpm using SHARK.

Summary

Text entry is an important part of any user interface. Keyboards are by far the fastest form of entry, but they are restricted to certain physical situations. For more nomadic text entry, buttons and pens can be used. These are all slower than the keyboard and in most cases the input is ambiguous. The input ambiguity is overcome by multiple entries, menu selection from the “best-N” after a preliminary entry or the use of a statistical bigram or trigram language model.

---

[1] Reference to an automata book

[2] Soukoreff, R. W., and MacKenzie, I. S. “Theoretical Upper and Lower Bounds on Typing Speeds Using a Stylus and Soft Keyboard.” *Behavior & Information Technology*, 14 (1995), pp 370-379.

[3] Jurafsky, D. and Martin, J. H. *Speech and Language Processing*, Prentice-Hall, New Jersey, 2000.

[4] Darragh, J. J., Witten, I. H., “The Reactive Keyboard.” *Cambridge Series on Human-Computer Interaction*, Cambridge University Press, Cambridge, England, 1992.

[5] Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction*, Lawrence-Erlbaum, Hillsdale, NJ (1983).

[6] Silfverberg, M., MacKenzie, I. S., Korhonen, P. “Predicting Text Entry Speed on Mobil Phones”, *Proceedings of the ACM Conference on Human Factors in Computing Systems, (CHI 2000)*, ACM, New York, NY (2000), 9-16.

[7] James, C. L., and Reischel, K.M., “Text Input for Mobile Devices: Comparing Model Predictions to Actual Performance,” *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2001),* ACM, New York, NY (2001), 365-371.

[8] Soukoreff, R. W., and MacKenzie, I. S., “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 2003),* ACM, New York, NY (2003), 113-120.

[9] Gong, J., and Tarasewich, P., “A New Error Metric for Text Entry Method Evaluation,” *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2006),* ACM, New York, NY (2006), 471-474.

[10] MacKenzie, S., and Soukoreff, W., “Phrase Sets for Evaluating Text Entry Techniques,” *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI 2003),* ACM, New York, NY (2003), 754-755.

[11] Gong, J., and Tarasewich, P. “Alphabetically Constrained Keypad Designs for Text Entry on Mobile Devices,” *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2005),* ACM, New York, NY (2005), 211-220.

[12] Guinness Book of World Records.

[13] Matias, E., MacKenzie, I., and Buxton, W., “One-handed Touch Typing on a QWERTY Keyboard,” *Human-Computer Interaction* 11. p 1-27.

[14] MacKenzie, I. S., Kober, H., Smith, D., Jones, T., and Skepner, E. “LetterWise: Prefix-based Disambiguation for Mobile Text Input,” *User Interface Software and Technology (UIST 01),* ACM, (2001), pp 111-120.

[15] Pavlovych, A., and Stuerzlinger, W., “Less-Tap: A Fast and Easy-to-learn Text Input Technique for Phones,” *Graphics Interface 2003*.

[16] Gong, J. and Tarasewich, P. “Alphabetically Constrained Keypad Designs for Text Entry on Mobile Devices,” *Human-Factors in Computing Systems (CHI 2005)*, ACM, (2005), pp 211-220.

[17] Ingmarsson, M., Dinka, D., and Zhai, S., “TNT – A Numeric Keypad Based Text Input Method,” *Human Factors in Computing Systems (CHI 2004)*, ACM, (2004), pp 639-646.

[18] Conrad, R., and Longman, D., “Standard Typewriter Versus Chord Keyboard: An Experimental Comparison,” *Ergonomics*, 8. (1965) p. 77-88.

[19] Lyons, K., Starner, T., Plaisted, D., Fusia, J., Lyons, A., Drew, A., and Looney, E.W., “Twidler Typing: One-Handed Chording Text Entry for Mobile Phones”, *Human Factors in Computing Systems (CHI 2004)*, ACM, (2004), pp 671-678.

[20] Wigdor, D., and Balakrishnan, R., “A Comparison of Consecutive and Concurrent Input Text Entry Techniques for Mobile Phones,” *Human Factors in Computing Systems (CHI 2004)*, ACM, (2004), pp 81-88.

[21] Wobbrock, J. O., Myers, B. A., and Rothrock, B., “Few-key Text Entry Revisited: Mnemonic Gestures on Four Keys”, *Human Factors in Computing Systems (CHI ’06),* ACM, (2006), pp 489-492.

[22] Wilson, A. D., and Agrawala, M., “Text Entry Using a Dual Joystick Game Controller”, *Human Factors in Computing Systems (CHI ’06),* ACM, (2006), pp 475-478.

[23] Partridge, K., Chatterjee, S., Sazawal, V., Borriello, G., and Want, R., “TiltType: Accelerometer-Supported Text Entry for Very Small Devices”, *User Interface Software and Technology (UIST ’02),* ACM, (2002), pp 201-204.

[24] Proschowsky, M., Schultz, N., and Jacobsen, N. E., “An Intuitive Text Input Method for Touch Wheels”, *Human Factors in Computing Systems (CHI ’06)*, ACM, (2006), pp 467-470.

[25] MacKenzie, I. S., and Zhang, S. X., “The Design and Evaluation of a High-Performance Soft Keyboard,” *Human Factors in Computing Systems (CHI ’99)*, ACM, (1999), pp 25-31.

[26] Zhai, S., Hunter, M., and Smith, B. A., “The Metropolis Keyboard – Exploration of Quantitative Techniques for Virtual Keyboard Design,” *User Interface Software and Technology (UIST ’00),* ACM, (2000), pp 119-128.

[27] Isokoski, P. “Performance of Menu-Augmented Soft Keyboards,” *Human Factors in Computing Systems (CHI ’04)*, ACM, (2004), pp 423-430.

[28] Masui, T., “An Efficient Text Input Method for Pen-based Computers,” *Human Factors in Computing Systems (CHI ’98)*, ACM, (1998), pp 328-335.

[29] Goldberg, D., and Richardson, C., “Touch-Typing With a Stylus,” *InterCHI ’93*, ACM, (1993), pp 80-87.

[30] Venolia, D., and Neiberg, F., “T-Cube: a Fast Self-disclosing Pen-based Alphabet,” *Human Factors in Computing Systems (CHI ’94),* ACM (1994), pp 265-270.

[31] Blinkenstorfer, C. H. “Graffiti,” *Pen Computing*, (January 1995), pp. 30-31.

[32] Wobbrock, J. O., Myers, B. A., and Kembel, J. A., “EdgeWrite: A Stylus-Based Text Entry Method Designed for High Accuracy and Stability of Motion,” *User Interface Software and Technology (UIST ’03)*, ACM (2003), pp. 61-70.

[33] Wobbrock, J. O., Myers, B. A., and Rothrock, B., “Few-key Text Entry Revisited: Mnemonic Gestures on Four Keys”, *Human Factors in Computing Systems (CHI ’06)*, ACM (2006), pp 489-492.

[34] Wobbrock, J. O., and Myers, B. A., “Trackball Text Entry for People with Motor Impairments” *Human Factors in Computing Systems (CHI ’06)*, ACM (2006), pp 479-488.

[35] Mankoff, J. and Abowd, G. D., “Cirrin: A Word-level Unistroke Keyboard for Pen Input,” *User Interface Software and Technology (UIST ’98)*, ACM (1998), pp 213-214.

[36] Perlin, K. “Quikwriting: Continuous Stylus-based Text Entry,” *User Interface Software and Technology (UIST ’98)*, ACM (1998), pp 215-216.

[37] Kristensson, P., and Zhai, S., “SHARK^2: A Large Vocabulary Shorthand Writing System for Pen-based Computers,” *User Interface Software and Technology (UIST ’04)*, ACM (2004), pp 43-52.