Chapter 6:
CASE STUDY 2:
CHORD KEYBOARDS

Introduction

For the most part, when we type on a conventional keyboard, each key is associated with a particular character or symbol – that which corresponds to the label on the key cap – and that symbol or character is entered by pushing a single key. In contrast, there is another class of keyboards, chord, or multi-press keyboards, one generally has to push down a combination of two or more keys in order to enter a character. We see a trivial example of this in the use of the SHIFT key when typing on a standard QWERTY keyboard. Here, to get an upper-case character, one must make a two-key chord consisting of the SHIFT key and the desired character key. What distinguishes the keyboards discussed in this chapter from this example is that chording is the norm rather than the exception, and frequently employs more than two elements in the chord.

One of the attractions chord keyboards is that for a given number of symbols or characters, they require fewer keys, and can therefore occupy less space. We can see a wonderful example of this by returning to our previous example of the QWERTY keyboard, and digging into its history.

Figure 1, shows two typewriters from the late 1890s. The first has no shift key, so it needs two complete sets of alphabetic keys: one for each of the upper-case and lower-case character sets. In contrast, the second typewriter has *two* distinct shift keys, labeled *Cap* and the other *Fig*.

Here is how this second unit works. Like many mobile phones today, there are 3 symbols associated with each character key: a lower and upper case character as well as a number or symbol. Which one you get when you type depends on if a SHIFT key is pressed at the same time, and if so, which one. As with typing today, pushing a key alone gives you the lower-case version of the indicated alphabetic character. Pushing that same key while holding down the *Cap* key, i.e., chording, types the upper-case character. Finally, pushing a key while holding down the *Fig* key results in the associated number or punctuation symbol being typed.

Figure 1: The SHIFT Key and Chording

Two late 19th Century QWERTY Keyboards. The upper one is the Caligraph – New Century 6, made in 1899 by the American Writing Machine Company, New York. The lower one is the Commercial Visible 6, made in 1898 by Commercial visible Typewriter Co., New York. While both use the QWERTY keyboard layout, note that the New Century 6 does not have a shift key. Rather, it has two QWERTY key sets: the upper one for upper-case, and the lower one for lower-case. In contrast, the Commercial Visible 6 has two shift keys: one to get the upper-case character associated with a particular key, and the other to get the associated special character (number or punctuation). (Photos by Martin Howard from his personal collection of antique typewriters[1].)

[1] For more information on Martin's phenomenal collection, see http://www.antiquetypewriters.com.

The net result of all of this: adding the Cap and Fig keys to the second keyboard reduced the overall size of the keyboard from seven to three rows of keys, compared to the first one.

> *Since the modern QWERTY keyboard does not have the Fig key, but rather a separate row for numerals, one might be lulled into thinking that we have landed somewhere in the middle of these two historical examples. Further analysis will show the contrary – we have gone farther, since in this light, we have at least 3 or 4 shift keys (SHIFT, CTL, ALT, FN, …) and even these shift keys are chorded (pushed together) in some cases (e.g., CT-Alt-Del). In this exercise, note that if we use the term "shift key" to describe this generic class of chorded modifier keys, it can help us see patterns and relationships that might help us in thinking more clearly in future design situations.*

While the previous example illustrates that even conventional QWERTY keyboards involve chording, nevertheless, we generally are not referring to them when we speak about chord keyboards. The distinction is essentially one of degree: what percentage of tokens entered involve chording as opposed to single key entry, and how many keys are involved in each chord?

Figure 2: A Trumpet Player "Chording" on Valves to Play Note (Photo: Roger Dannenberg)

A comparison between a piano and a trumpet reveals subtle distinction, perhaps worth filing in the back of one's mind. Most would say that the piano is a good example of a chording keyboard. After all, more often than not one is playing more than one note at a time, and when clustered together, these are even called chords. On the other hand, since the trumpet can only play one note at a time, it would rarely be considered a chording device. Yet, I would argue that it has more in common with the chording keyboards that we are discussing in this chapter than does the piano.

While, unlike the piano, the trumpet cannot play chords, the three finger-operated valves that make up its "keyboard" are heavily used in combination in order to play the instrument (see **Figure 2***). There are seven ways in which the three valves can be combined – eight if you include the case where no valve is depressed), which establishes that that the trumpet is, after all, a chording device –its self a reminder to be open-minded as to what we consider a "keyboard".*

But the distinction that I really wanted to point out is that the piano is dissimilar from the chording keyboards that we discuss in this chapter because despite the ability to play a chord, each note played is the result of the striking of only one key, whereas with the trumpet the note played is a function of what combination of valves is depressed (the "chord" is defined in terms of input rather than output), as well as the player's breath and embouchure control (but that is another story).

Finally, just to reinforce the need to look at the bigger picture, let me contradict myself and note that the piano is like the QWERTY keyboard if one takes into account its pedals (typically una corda, sostenuto and sustain), for these are very analogous to the QWERTY keyboard's shift keys. They just happen to be in a different form factor and are operated by the foot rather than the hand.

In this chapter we are mostly going to consider keyboards that have few keys, where a rich combination of keys is used, and where the result of each chord is a single *token*[1]. Chord sets that are used to replace conventional keyboards, for example, typically have between six and eight keys per hand (we will see one and two handed versions).

[1] I am being careful with my language here so as not to overly limit the implications of what I write. In cases of typing text, by *token* the typical meaning would be "alphanumeric character". However, limiting ourselves to that is overly restrictive and misses potentially powerful design opportunities. As we shall see, for example, the chords on the keyboards used for court reporters typically specify higher information entities, phonemes, rather than graphemes (letters). Likewise, one should not eliminate the possibility of using chords to invoke user-defined macros, such as having the chord "sy" on a QWERTY keyboard function as an abbreviation for "Sincerely yours". Even if not used for typing, chording may be appropriate for entering other entities – things like specific functions, or even trumpet notes, for example.

How Many Symbols from How Few Keys?

Normally you will read that non-chording keyboard consisting of *n* keys can produce a maximum of *n* distinct outputs. Likewise, you will read that the maximum of distinct outputs if it were a chording keyboard would be 2^n-1.

While this is kind of true, there are a few things that are important to keep in mind.

1. Don't just look at the keyboard: How many fingers does the operator have to use? No matter how many key combinations there are, if the operator is restricted to one hand, for example, only chords consisting of 5 or less *simultaneously reachable* keys should be considered.

2. Number of key states: The rule generally given assumes that the keys only have two states, i.e., up and down. But as we shall see, some chord keyboards have buttons that have more than two states. Consider a 2-key chord keyboard, where each key has three states:

   I. Middle (M): the default position. The position that the key assumes when not being touched.

   II. Forward (F): the state when the key is titled in the forward position. When released, it springs back to M.

   III. Back (B): the state when the key is tilted back. When released, it springs back to M.

   In this case these two keys enable the following number of states:

   M_1 M_2
   M_1 F_2
   M_1 B_2
   F_1 M_2
   F_1 F_2
   F_1 B_2
   B_1 M_2
   B_1 F_2
   B_1 B_2

   Assuming that no symbol is being triggered in state , that gives us 8 distinct symbols from only two(3-state) keys. The general rule, then, is that the number of possible combinations of n keys is S^n-1, where S is the number of states of the keys. Thus, for example, the full 26 letters of the alphabet could be articulated with 3 ternary keys instead of 5 binary keys.

3. What triggers the chord is important: The design of chord keyboards has to assume that all of the keys in a chord will never be pushed at exactly the same time. Despite being a "chord" keyboard, sequencing may be important. To the extent that it is a chord keyboard, the SHIFT key on a QWERTY keyboard has to be depressed before the character key. On the Writehander keyboard (Figure 14), the chord transmitted is determined by what buttons operated by the 4 fingers are depressed at the time one of the thumb-operated buttons is depressed – which thumb key also contributing to the make-up of the chord.

4. Remember the trumpet: Don't assume that the transmission of the chord is triggered by pushing the buttons. With the trumpet, the note is triggered by blowing, so the state where all buttons (i.e., valves) are up is also a distinct state. Sometimes this may be a parameter that can be used in your design.

In general, there are two key attributes of chord keyboards, each the flip side of the other:

1. If you can type, you can touch type.

2. If you can't touch type, you can't type.

With conventional keyboards, the labels on the key-caps enable novices to hunt-and-peck. Since any single key on a chord keyboard can be involved in entering a number of different characters – depending on what other keys are pushed – there is generally no comparable labeling scheme possible.[1] Hence, the ability to hunt-and-peck is pretty much eliminated.

This significantly raises the bar of entry compared to conventional keyboards: you pretty much have to have memorized the chords and the keyboard before you can use it at all. On the positive side, learning to touch type on a chord keyboard is typically orders of magnitude faster than with a QWERTY keyboard (the learning speed, that is, not likely the typing speed). One of the reasons for this is that the fingers generally do not move from key to key, and more-or-less remain in "home position". Thus, an investment of one to two hours is not atypical of what is required to acquire basic touch-typing skill. But just remember: even if you are fine 99% of the time, if you forget the chord for some special character, you had better have some documentation close at hand.

Chord Keyboards have two other important attributes. First, they can afford one-handed typing[2]. This is a benefit to disabled users (Kirschenbaum, Friedman, & Melnik, 1986), as well as those who want to occupy the other hand in some other task, such as pointing with a mouse. Second, unlike stylus driven input and conventional keyboards, one-handed chord keyboards are one of the only manual text entry methods that can be undertake while mobile or in the presence of to vibration, such as when walking, or in a bumpy train (as an extreme example, just consider the conditions under which you can type compared to where a trumpeter can play their instrument)[3]. Their potentially smaller size and ability to be operated while ambulatory are two key reasons why their study is relevant to those interested in compact portable devices.

People have worked with and explored the potential of chord keyboards for a number of years. Conrad and Longman (1965) state that the first reported example of a chord keyboard dates from 1942. An extensive review of the literature is given by Martin (1980), and a shorter version under her married name, Noyes (1983b). Seibel (1972) gives a good survey of relevant human-factors studies. These investigate issues such as the effect of number of buttons, number of chords, number of hands, and the relative difficulty of different combinations. But the one thing that all of these studies miss is how far back chord keyboards actually go – almost 100 years prior to the date reported by Conrad and Longman: 1857.

The first working chord keyboard, and therefore one of the first working typewriters (although the name had not been coined at the time) was *Livermore's Permutation Typograph* or

---

[1] As the old saying goes, never say never. As we shall see later in our discussion of *Self Revelation*, there is at least one design that partially addresses this issue. Having recognized the problem is perhaps the first step in developing some useful hardware or software innovation that helps address it in whole or in part.

[2] They are not unique in this capability but arguably better suited in most cases,

[3] The opposite extreme, a one-button keyboard, such as one using Morse code, is another example.

Pocket Printing Machine, invented by Benjamin Livermore of Hartland Vermont (Legrand, 2008). This device was about the size and shape of a deck of cards, and one typed using six keys, laid out in two rows at one end, as illustrated on the cover of the1857 booklet describing the device, shown in Figure 3. The permutation typograph is reported to have contained a 20-foot roll of thin paper onto which what was entered on the keyboard was printed. And, consistent with what I said about chord keyboards being relevant for portable applications, and eyes-free typing, Livermore – the inventor – generally kept his in his pocket and took notes on it as he walked around. Livermore is clearly the unknown pioneer who cut the first path in the frontier leading to the text input mechanisms used by today's wearable computer community (Lyons, et. Al, 2004a, 2004b) and texting on mobile phones!

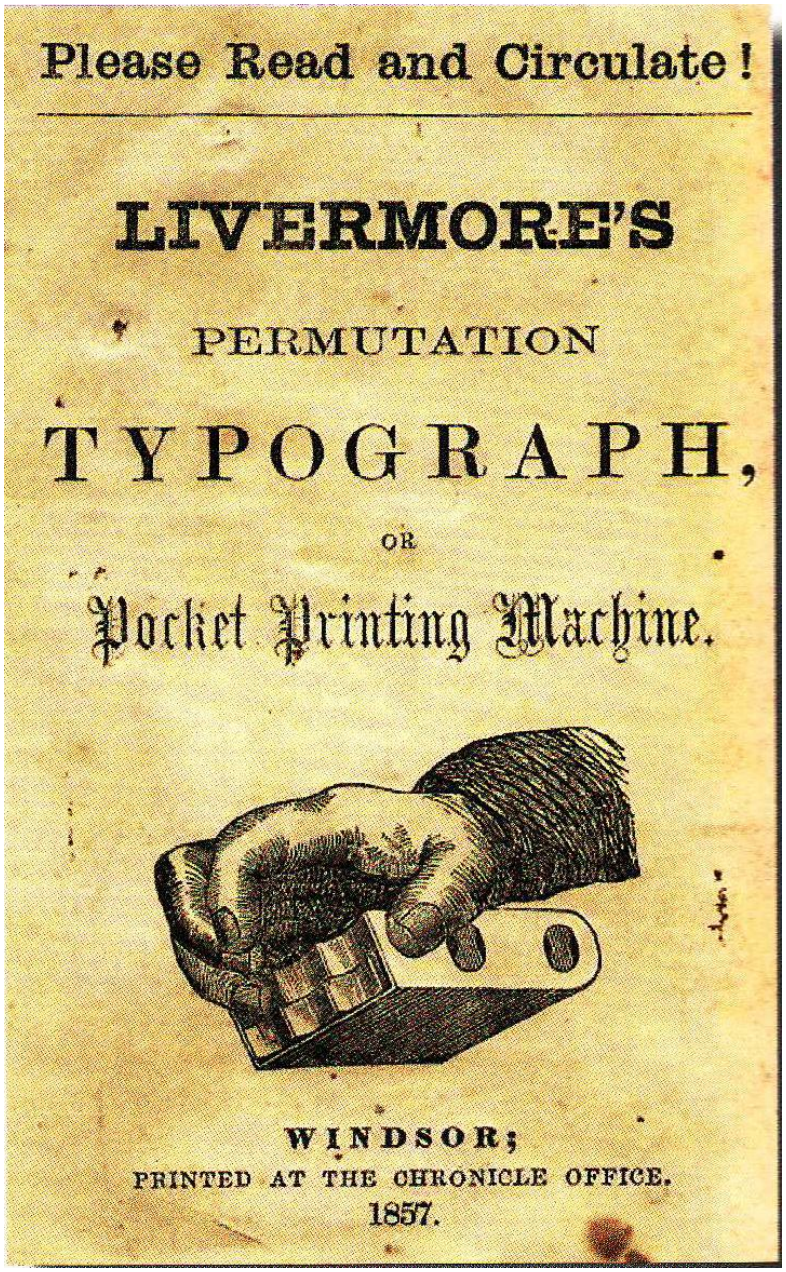

Figure 3: The First Known Chording Keyboard Typewriter

Figure 4: The PARCtab

This is a prototype handheld device called the PARCtab, developed at Xerox PARC in 1992 and built in 1993. I was responsible for the concept of the basic form-factor, in particular the buttons (which supported chording). Note the similarity to the trumpet. That was not a coincidence. The idea was to have a hand-held device for which the buttons could be operated by the same hand that held it – something very rare, even today. But what I like even more is the similarity to Livermore's Typograph (Figure 3) which I did not know about at the time. Why did we come to the same basic solution? Because both designs derived from the same properties of the human hand, physical and kinematic. The technology may have changed, but human physiology has not. (Photo: Liz Russ)

The Braille Keyboard

While Livermore's Permutation Typograph is interesting today due to it being a "mobile" device, in terms of being a chord keyboard, the seeds for the first chorded writing device were planted almost 25 years earlier with the development of Braille, a writing/reading system for the blind. Developed by a Frenchman, Louis Braille, the system enables one to read with one's finger tips, rather than eyes, by sensing patterns of raised dots embossed in paper.[1]

[1] In so doing, Braille unwittingly introduced the first binary code for representing text, one that preceded Morse Code, another early encoding, by about 10 years.

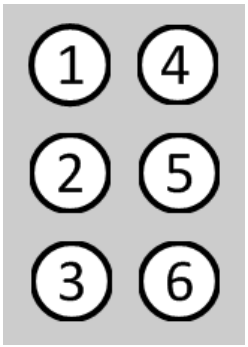

Figure 5: Six-dot Braille Cell

The cell dots are numbered in a specific order. The specific input intended by a cell is determined by the absence or presence of each of the six dots.

The dots are grouped into cells consisting of two columns, traditionally, of 3 dots each (Figure 5). Having only six dots, a single cell can only represent $2^6 = 64$ distinct entities (a blank cell represents a space). While this is sufficient for the alphabet, it is not sufficient to capture all of the special characters, numbers and punctuation found in English, or on a standard QWERTY keyboard, for example. Hence, to represent some characters, two cells are required.

Figure 6: Perkins Standard Brailler

One types using the row of nine keys on the front of the device. The wide centre key is the Space key, which is operated by either thumb. Depressing it enters a blank cell. Each of the three keys on either side of the Space key are for entering one of the six dots in the cell. They are operated by the pointing, index and ring fingers of the left and right hand, respectively. To grasp how the fingers map to the dots, place your hands as if you were about to clap, with your thumbs pointing up. Then the three fingers used by each hand are in the same position of the corresponding dots. Hence, your left and right pointing fingers enter dots 1 & 4, respectively, index fingers 2 & 5, and ring fingers 3 & 6. The dot keys for the desired character are pushed simultaneously, hence the chording.

Keyboards, such as the one illustrated in Figure 6, were developed to enable one to efficiently type Braille on paper, and later, into computers. Since it was easy to map the three dots of the left column to three fingers of the left hand, and those of the right column to three fingers of the right hand, these keyboards understandably enabled a complete cell to be entered as a single chord (as described in the Figure caption).

Even before the advent of computers, Braille keyboards provided an example of one of the key trade-offs in using chording for larger symbol sets: having a low number of keys and therefore requiring compound chords for some characters, *versus* having chords with a larger number of keys, and thus being able to enter each symbol with a single chord.[1] This trade-off is the source of much innovation and ingenuity, especially given the frequent desire to make a one-handed chording keyboard.

The NLS System: Engelbart and English

In the realm of human-computer interaction, the NLS system developed by Engelbart and English (1968) is important to our discussion. This is the project that introduced the mouse, and therefore had a huge impact on multiple generations of computer users. What is easily missed, however, is that the goals of this work had nothing to do with improving access to computing for casual users or making computers easy to use. Rather, they were designing a system with which highly trained operators could reach their full potential. By analogy, they were designing a violin for a virtuoso performer, rather than *Guitar Hero*[2] for the amateur. It is somewhat ironic, therefore, that the mouse was adopted by novices far faster and enthusiastically than by the power-users targeted by the NLS project. For a long time, they clung to keyboard-based command-line interfaces.

Stepping back, from the perspective of input, the NLS system pursued a bi-manual approach that employed three main devices:

1. A conventional QWERTY keyboard
2. A three-button mouse
3. A five-button chord Keyboard.

These are illustrated in Figure 7.

[1] As it turns out, that there are versions of Braille that have added an additional dot to each column, so-called eight-dot Braille, thereby expanding the number of distinct symbols that can be represented by a single cell. For additional background on this, see Dixon (2007).

[2] Guitar Hero refers to series of music video games, first released in 2005, in which the game controller takes the shape of a musical instrument, initially a guitar, which one “plays”. The look of the instrument and the sound that result resembles the real thing. The expertise required to “play” the “instrument” does not – not by a long shot.

Figure 7: The NLS Input Devices

This image is a frame from the film of the "Mother of all demos", the demonstration of Engelbart and English's work that accompanied their classic 1968 paper at the Fall Joint Computer Conference.[1] *Left-to-right one sees the left hand on the chord keyboard, the QWERTY keyboard in the middle, and the mouse in the right hand. Note that the mouse is not the original one-button wooden mouse shown in Figure 5 of Chapter 2. Rather, it is a three-button mouse – a fact which is important to our discussion.*

[1] http://sloan.stanford.edu/mousesite/1968Demo.html

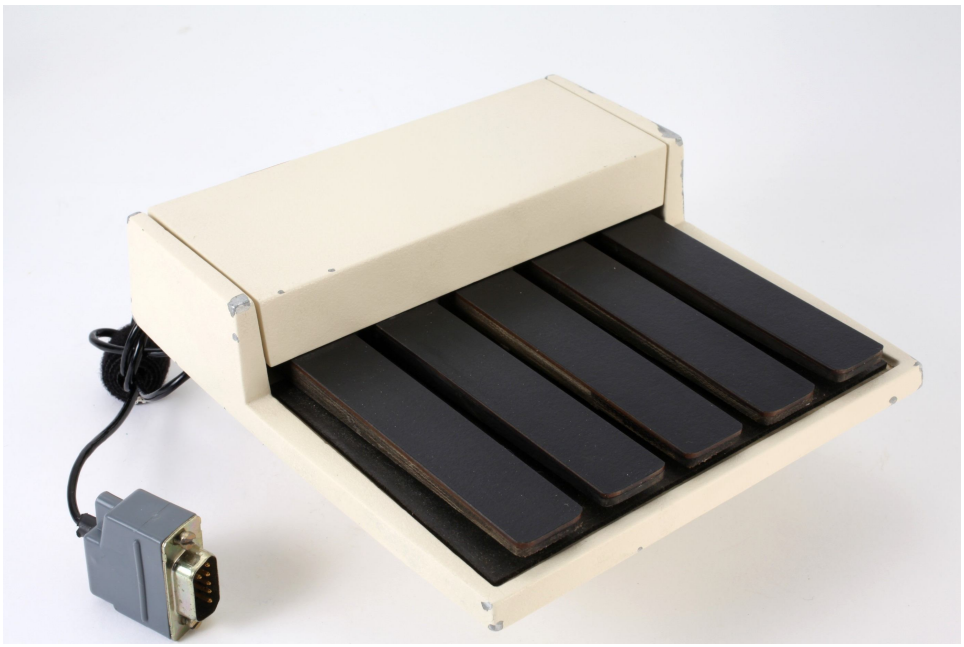

Figure 8: The Xerox PARC Version of Engelbart's 5-button Chord Keyboard

Around 1973 Xerox Palo Alto Research Center (PARC) began building 5-button keyboards to use with their experimental in-house workstations, starting with the Alto. As was the accompanying mouse, these were very much com based on those used with Englebart's NLS system.

The use of the mouse and QWERTY keyboard was similar to that common to the graphical user interfaces that have been around since the early-to-mid 1980s. The mouse was typically operated by the dominant hand and used for spatial tasks such as graphically pointing and selecting. The QWERTY keyboard was used for sustained text entry. It is in the five-button chording keyboard that we begin to encounter something less familiar. Since there are a few surprises here, I'm going to go into a bit of detail about its use.

The chording keyboard was there to address a very specific issue. If entering text was frequently interleaved with using the mouse, the user needed to either bring the mouse hand back and forth to the keyboard to type, or, type with the non-mouse hand only. As most computer users know, this can be done, but only with a penalty in performance time.[1]

The 5-key piano-like chording keyboard was developed to provide an alternative means of entering text – one that did not require the mouse hand to move to the QWERTY keyboard. Rather, one could enter text with one hand on the mouse and the other on the chording keyboard, as seen in Figure 7.

In describing this technique, Engelbart & English (1968) refer to it as "one-handed typing." This description has given rise to the mistaken belief that typing in this mode employed only the chord keyboard. Yes, many things could be typed using just that device. But so to can a violinist play music using only three of the four strings on the instrument. While some music

[1] One handed typing on a QWERTY keyboard is simply much slower than using two-handed. And moving a hand back-and-forth between mouse and keyboard incurs a penalty in terms of *homing time*, as will be seen in our discussion of the *Keystroke-Level Model* in the next Chapter.

can be played, the full repertoire cannot. Likewise, to access the full character set, this mode of entering text required seven keys, just as the violin needs four strings.

Five binary keys mean that only 2^5-1 = 31 different characters can be entered. So, as he designed it, yes, Engelbart could enter the 26 lower-case letters of the alphabet and five other characters: comma, period, semicolon, question mark, and SPACE using just the five keys of the chord keyboard.

However, to enter the rest of the character set - such as the upper-case letters of the alphabet, digits, numerical operations, and additional punctuation - two more buttons were required. These were provided by the left and middle buttons of the three-button mouse.

In effect, Engelbart used a virtual 7-button chord keyboard split over two different physical devices. This gave him access to a full repertoire of 2^7-1 = 127 different characters. The question is, how did he map the chord combinations onto the character set?

The short answer – at least for those with some background in computer science – is that he used a clever variation of 7-bit ASCII. Figure 9 will help provide a more complete explanation.

The main table in the Figure has 32 rows.

- Each of the 32 rows in the main table represents one of the unique combinations in which the five buttons of the chord keyboard can be depressed.
- Each of the five left-most columns corresponds to one of the five buttons on the chord keyboard.
- A “1” in a cell indicates that the key associated with that column is depressed in the chord associated what that specific row. For example, the key associated with the left-most column is only depressed in the chords represented by rows 17 to 32. Likewise, the top row indicates the situation where none of the five keys are depressed.
- If only the chord keyboard is used, the character that is entered for any of the 31 possible chords (rows 2 –32) is shown in the corresponding row in the 6th column, labeled “State 1”.
- If the middle mouse button is depressed, the character entered is determined by which, if any, of the 5 chord keyboard buttons are also depressed, and is indicated in the corresponding row in the 7th column, labeled “State 2”.
- If the left mouse button is pushed, it works the same way, except the character entered is indicated in the corresponding row in the column labeled “State 3”.
- Finally, if both the left and middle mouse buttons are depressed simultaneously, then the character is likewise determined by which, if any, of the 5 chord keyboard buttons are depressed, and indicated in the corresponding row in the column labeled “State 4”.

Chord Keyboard — 2 Mouse Buttons

	bit 5	bit 4	bit 3	bit 2	bit 1	state 1	state 2	state 3	state 4
1	0	0	0	0	0		Cmd Abort		
2	0	0	0	0	1	a	A	!	show one level deeper
3	0	0	0	1	0	b	B	"	show one level deeper
4	0	0	0	1	1	c	C	#	show all levels
5	0	0	1	0	0	d	D	$	show top level only
6	0	0	1	0	1	e	E	%	current statement level
7	0	0	1	1	0	f	F	&	recreate display
8	0	0	1	1	1	g	G	'	branch show only
9	0	1	0	0	0	h	H	(	off
10	0	1	0	0	1	i	I	)	show content passed
11	0	1	0	1	0	j	J	@	I or k off
12	0	1	0	1	1	k	K	+	show content failed
13	0	1	1	0	0	l	L	-	show plex only
14	0	1	1	0	1	m	M	*	show statement numbers
15	0	1	1	1	0	n	N	/	hide statement numbers
16	0	1	1	1	1	o	O	^	frozen statement windows
17	1	0	0	0	0	p	P	0	frozen statement off
18	1	0	0	0	1	q	Q	1	show one line less
19	1	0	0	1	0	r	R	2	show one line more
20	1	0	0	1	1	s	S	3	show all lines
21	1	0	1	0	0	t	T	4	first lines only
22	1	0	1	0	1	u	U	5	normal refresh display
23	1	0	1	1	0	v	V	6	inhibit refresh display
24	1	0	1	1	1	w	W	7	all lines, all levels
25	1	1	0	0	0	x	X	8	one line one level
26	1	1	0	0	1	y	Y	9	blank lines on
27	1	1	0	1	0	z	Z	no-op	blank lines off
28	1	1	0	1	1	,	<	[	no-op
29	1	1	1	0	0	.	>	]	no-op
30	1	1	1	0	1	;	:	↑	no-op
31	1	1	1	1	0	?	\	ALT	centerdot
32	1	1	1	1	1	SP	TAB	CR	no-op

State 1	state 2	state 3	state 4	
11	10	01	00	ASCII
00	01	10	11	Engelbart

All State 2, 3 & 4 entries required mouse button[s]as well as well as chord keyboard to specify.

Figure 9 Engelbart and English Chord Keyboard Encoding Scheme

This table shows how the 5 buttons on the chord keyboard and 2 mouse buttons were used to enter text. (Based on Engelbart, 1973).

To wrap up the explanation of how the buttons of the chord keyboard and the mouse relate to the table:

- The illustration on the Chord Keyboard at the bottom right of Figure 9 indicates which bit (1-5) is associated with which key, and therefore which key corresponds to each of the first five columns in the table.
- The illustration of the mouse in the middle right of the figure shows the labelling of the buttons (1-3), and which of the seven bits (6 & 7) is mapped to which button. These two bits do not map directly into columns in the table, hence the next point.
- For the readers who are computer scientists: the small table at the top on the right-hand side shows how Engelbart remapped the meaning of bits 6 and 7, compared to 7-bit ascii. This was clever, since it gave him access to the lower-case alphabet using only the 5 least significant bits, i.e., using just the buttons on the chord keyboard.

Using the limited character set available with just the 5-button chording keyboard, Engelbart was reported to have been able to achieve a typing speed of 35 words per minute with his right hand, and 25 words per minute with his left. It is also reported that it took him about 10 hours to reach 10 words per minute (Noyes, 1983).

Remember, the speeds reported above included only 31 of the 127 of the full character set. His typing speed would have been slower if he was employing both hands, and it would take far more than 10 hours to learn them all. However, there was always the option to revert to the QWERTY keyboard – which was frequently done. As stated in Engelbart and English (1968):

- *One-handed typing with the handset is slower than two-handed typing with the standard keyboard. However, when the user works with one hand on the handset and one on the mouse, the coordinated interspersion of control characters and short literal strings from one hand with mouse control actions from the other yields considerable advantage in speed and smoothness of operation.*
- *For literal strings longer than about ten characters, one tends to transfer from the handset to the normal key board.*
- *Both from general experience and from specific experiment, it seems that enough skill to make its use worthwhile can generally be achieved with about five hours of practice. Beyond this, skill grows with usage.*

Given the historical importance of this system, I have gone into so much detail about the text entry because it is so difficult to extract from the literature. By the same token, the exercise helps build a stronger sense of how important such details are in terms of the ultimate user experience. No matter how good any of the chord keyboards discussed look or feel mechanically, the complexity of the encoding scheme – how long it takes to learn, the proneness of error, or the speed of entry, may dominate the value.

Not all chord keyboards are designed as alternatives for QWERTY keyboards, or for entering large difficult to remember symbol sets. One might just want to design a 3-button mouse in such a way that – by chording – it is able to function as a virtual 2^3-1=7 button mouse. An example of how the industrial design of the mouse can facilitate one doing so can be seen in the next example, the DePraz mouse from 1980, shown in Figure 10.

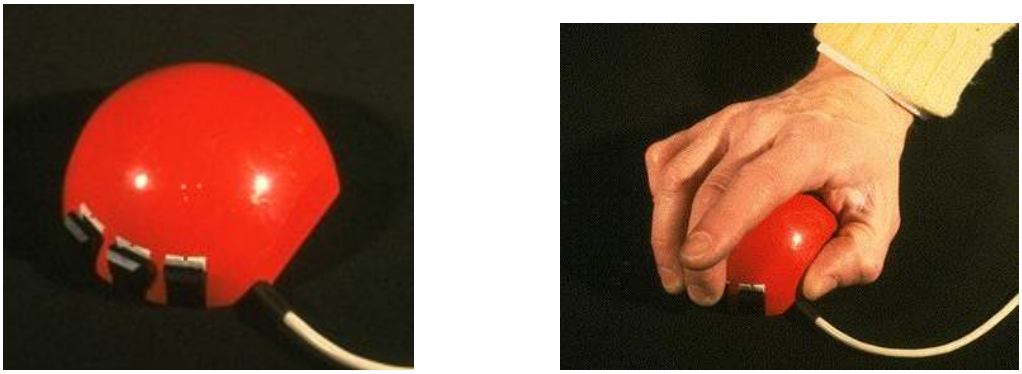

Figure 10: Mouse Button Layout to Facilitate Chording.

The Depraz or "Swiss" mouse, from 1980. The turtle-back hemisphere design affords holding the mouse and laying the fingers on the three-buttons in a manner not unlike how one grips a trumpet, thereby enabling control of each button by a different finger, and hence, chording.

In this case, the mouse is shaped like a turtle's back, which encourages the device to be held in what is generally called a *power grip.* This is not unlike the pose of the trumpeter's hand, and, likewise, facilitates each of the three middle fingers simultaneously laying over, and independently operating, each of the three mouse buttons.[1]

But what if one wants to go beyond the seven distinct functions that three chording mouse buttons might provide? As we shall see in later chapters, one approach would to use the mouse to make gestures as well as for pointing.

[1] While facilitating chording, this approach was not without its draw-backs. First, the power-grip generally resulted in a larger bend in the wrist than more conventional designs, and this in turn meant that pointing was done more with the larger muscle groups of the wrist and arm, rather than the smaller muscle groups of the fingers and wrist used by most other mice. This negatively impacted fine motor control. The other issue was that the motor action in activating the mouse buttons was parallel to the plane of the pointing motion, rather than more-or-less orthogonal to it – as was the case in most other mice. Hence, there was potential for button activation to interfere with pointing.

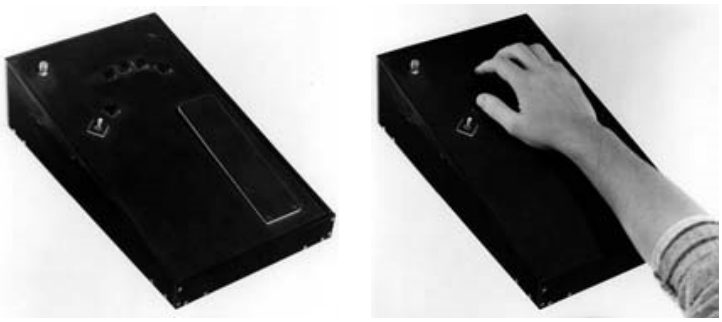

Figure 11: NRC of Canada Chording Keyboard

This keyboard was used for entering musical note durations and bar lines using the left hand while the right hand (using two thumb wheels not shown) was used for specifying pitch and entry point. Note the palm bar. All other keyboards discussed in this chapter use only the fingers for key entry.

Another early two-handed system that used one hand on a chording keyboard and the other on a pointing device was developed in between 1967 and 1969 at the National Research Council of Canada. To study human-computer interaction, an interactive system for music composition and a system for animation were developed (Pulfer, 1971; Wein, 1990). With the music system, for example, the right hand was used with two thumb-wheels to determine where in pitch and time a note was to be entered, and a chording keyboard (Figure 11) was used by the left hand to enter the note duration.

Chord keyboards and mice were widely available at Xerox's Palo Alto Research Center through the 1970s. The keyboards were based on Engelbart and English's "piano-like" design. They did not, however, achieve great popularity[1]. However, that chording keyboards never caught on is not proof that they cannot provide powerful solutions to some user interface problems. The system for entering music discussed above (Pulfer, 1971) is one example. Furthermore, as Seibel (1972) points out, the fastest rates of keyed input have been achieved using chording keyboards.

---

[1] Dan Sweinhart of PARC recently made the following comments about the demise of the chord keyboard at PARC:

I always liked the chord keyboard - got pretty good at it. But there are some disadvantages, which I believe led to their abandonment at PARC/Xerox:

1) You have to learn how to use them. Teaching a new form of typing before a system could be used effectively was considered too large a start-up transient for most customers to learn. It was a marketing, and perhaps a human factors issue.

2) The chordset was originally used, along with the mouse, both to issue commands (such as D-W for "delete word") and to enter small amounts of text. The editors in place at the time switched from mode to mode depending on the commands that had been issued, so that commands and text could be distinguished. When we switched over to the modeless, direct-manipulation style, the chordset was typically used simply for encoding various editing operations. The QWERTY keyboard had to be used for text. Users were forced to revert to the current style of switching back and forth from keyboard to mouse/keyset, and the value of the chordset faded away - function keys are easier to provide, and just as convenient.

SOME KEY ISSUES

With chord keyboards, speed of operation, proneness to error, and speed of learning are affected by a number of parameters:

- Physical Design: their number of buttons, their layout, and action.
- Size of Alphabet: how many symbols must be remembered
- Semantic level of symbols in alphabet: does a symbol represent an alphabetic character, a word, sentence ... ?
- Encoding scheme: the pattern assigned to each member of the alphabet
- Discoverability: Is the existence of the technique and/or how to use it self-revealing or discoverable?:
- Handedness: One handed or two handed (physically and cognitively)? If one handed, is it for the dominant or non-dominant hand, or can it be used equally well by either hand?
- Interference: cross-interference between chording and some other task

This list is not exhaustive, nor are its points mutually exclusive. They do, however, provide the basis for focusing our discussion.

PHYSICAL DESIGN

The symmetry of Engelbart and English's keyboard permitted it to be physically used by either hand.

In contrast, some devices are designed to fit the specific shape of the left or right hand. One example is shown in Figure 12. Designing the physical ergonomics to the physiology of the hand can improve performance and comfort. (See Eilam, 1989, for example.) However, this same tailoring of the form also means that, in contrast to the piano-like keyboard shown in Figure 7, the device can be only used by the hand (left or right) for which it was designed.

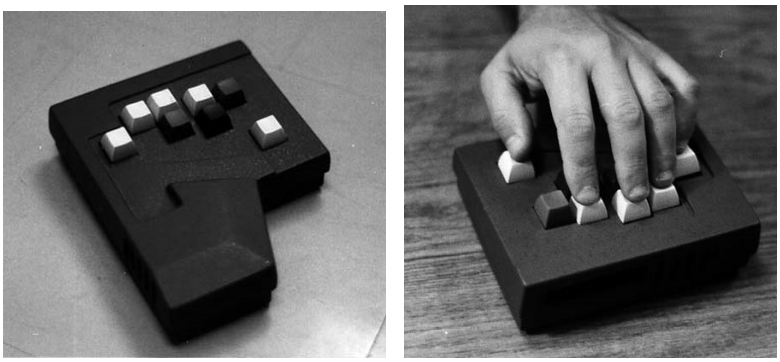

Figure 12: The Ergoplic Octima Chord Keyboard

The Octima is an example of an asymmetric chord keyboard. A left- and right-hand version was made. The form factor of each was designed to match the asymmetries of the intended hand. This is to improve the ergonomics of the device. Unlike the keyboard shown in Figure 8, the device can only be used by one hand.

This lack of interchangability can be a real issue. Consider the keyboard shown in Figure 13, the *Microwriter.*[1] This is a portable word processor that uses a chord keyboard whose design only affords right-handed text entry. However, the device has an RS-232 interface that permits it also to be used as the keyboard for a general-purpose computer. Since most right-handed people use their mouse In this case, it is quite conceivable that one may want to use a mouse in the right hand for pointing and the left hand to enter text using the chord keyboard. This is the way that Engelbart used his chord keyboard (**Error! Reference source not found.**). With the Microwriter this is not possible. The same aspects of its design that makes it well suited for the right hand prevents its use by the left.

Figure 13: Key Layout Preventing Left-Handed Operation (Microwriter Ltd.)

Another example of an asymmetrical design is the "Writehander" (Owen, 1978; NewO, 1978), shown in Figure 14. This device consists of four buttons mounted on a hemisphere so as to lie under the fingers. Eight other buttons lie within range of the thumb. The hemispherical shape of the device is worthy of note in that it takes advantage of the hand's ability to squeeze using a "power grip," much like a baseball. (The DePraz mouse[2] shown in Figure 10 also uses this form, although its three-buttons are mounted symmetrically, so that it can - in principle - be used by either hand.)

[1] Microwriter Limited, 31 Southampton Row, London WC1B 5HJ, England.

[2] Available from Logitech, 165 University Ave, Palo Alto, CA, 94301

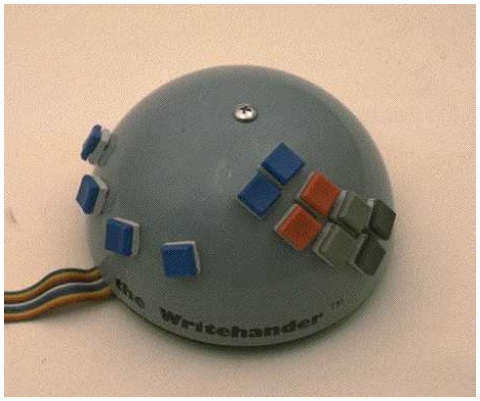

Figure 14: Convex Formed Chord Keyboard. (NewO Company)

The reason that the Writehander and Microwriter can only be used by one hand is the positioning of the keys operated by the thumb. To achieve a keyboard that was not symmetrical, but which could still be used by either hand, Rochester, Bequaert and Sharp (1978) developed a chord keyboard on which the position of the thumb keys was adjustable. (See also Bequaert & Rochester, 1977). This is shown in Figure 15.

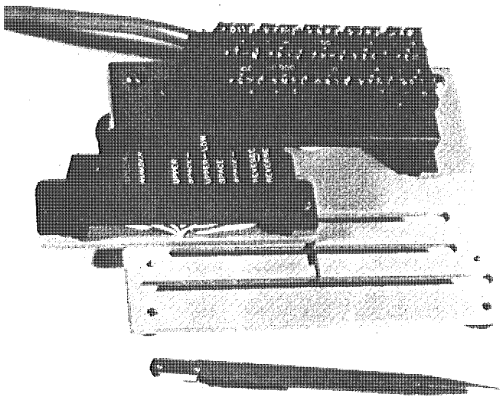

Figure 15: **Keyboard Adjustable to Either Hand (from Rochester et al.,1978)**

The number of keys on the keyboard is an important consideration. If there is only one key per finger, and the layout is appropriate, the hand can always remain on "home-row" (since it is the only row). Thus, if you can use them at all, you can touch-type. However, resting the fingers on the keys makes the problem of button quality (avoiding false key depressions), and their placement, critical. If you can rest your fingers on the keys without false depressions, then there is the danger that the key action will need too much force. On the other hand, if light action is desired, there is the danger of unintended depressions.

Key roll-over also presents a problem. What, for example, is the dividing line between two rapid sequential depressions and a sloppy chord? Among other considerations, this parameter may need to vary depending on the expertise of the operator. With the Writehander, the problem is side-stepped by having the chord transmitted upon depressing one of the thumb keys. The problem with this is that every chord must involve the thumb. Hence your ability to experiment with encodings is quite restricted.

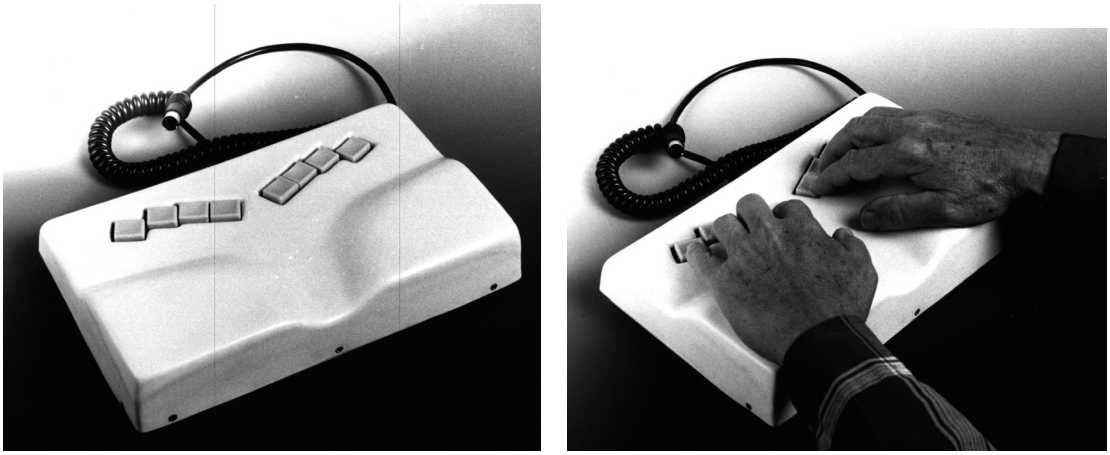

Figure 16: The Accukey Keyboard

This two-handed keyboard uses 3-state keys that move forward and backward from a neutral central position. (Photos: Vatell Corp.)

The *Accukey* keyboard from Vatell Corp. (Kroemer, Fathallah & Langley, 1988), shown in Figure 16 is unique in that it uses three-state keys. The design utilizes a two-handed eght button keyboard. Each chord is a two-button combination. However, instead of the two up/down states of conventional keys, the keys move forward and backwards from a neutral middle position. Using this approach, consistent fingerings are used for a given character, for example, and the direction of motion of the keys determines which mode (function, shift, alt or control) is used. McMulkin (1992) presents a study of the learning curve of five users of this keyboard over 60 hours of use, using a limited 18 character vocabulary.

Figure 17: Stenograph Keyboard

This chord keyboard is used in North America for court reporting. Speeds of over 200 words per minute can be achieved, but it takes about three or more years to achieve this level of performance.

Alternative designs for different classes of tasks should not be forgotten. Seibel (1962) developed a data entry station, "Rapid Type", that demonstrated the use of a modified QWERTY keyboard to enable chording input. By adding two extra shift keys, 150 common

words became available by chording. His data suggest that keying rates can improve by up to 150% by using the technique.

A version of this technique, called *RapidRiter,* was introduced commercially but is no longer in production.[1] The system was also based on a conventional keyboard, so users could type normally using their existing skills. The speed improvements came by permitting users to define abbreviations for frequently typed words or phrases. For example, holding down the "s" and "y" keys together could be an abbreviation for "sincerely yours," or "b" and "w" could be an abbreviation for "best wishes."

There are three main differences between this approach and most of the other designs discussed in this chapter:

The technique is not strictly chording. Chords supplement what one already knows (assuming one is familiar with the QWERTY keyboard).

- It builds upon the existing installed base of keyboards and skilled operators.
- The chord abbreviations are defined by the user. Hence, they are introduced gradually, and the issue of learning some predefined chord set is avoided. Of course if they are personalized, only that user can take advantage of the accelerators.

It is unclear why this product failed in the marketplace.

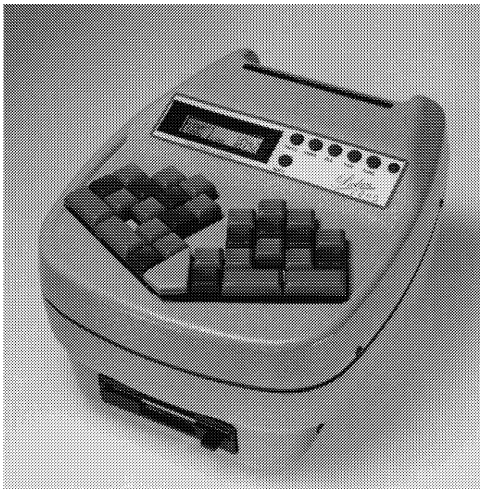

Figure 18: A Palantype Keyboard for Verbatim Transcription

This keyboard is. used in the United Kingdom for court reporting. It uses chorded key presses and highly trained operators can achieve data entry rates of 200 wpm or more.

Finally, chording keyboards are commonly used verbatim transcriptions of speech. This requires transcription speeds of 180 words per minute, or greater, which is about two to three times typical typing speed. In North America, a *Stenograph* machine, such as that shown in Figure 17 is used for this, while in the United Kingdom, a device called the *Palantype*, (Figure 18) is employed. The high data entry speeds accomplished with such devices is commonly cited as evidence of the high bandwidth possible with chord keyboards. However, it is important to realize that in order to achieve such speeds, the operators must often train for three to six years. Furthermore, what is being keyed in during the transcription process is a

[1] Quixote Corp., East Wacker Ave., Chicago, Illinois, USA 60601. tel: (312)-467-6755

phonetic script which must be transcribed into normal running text. Until recently, this transcription had to be done manually. Nowadays, this can be accomplished using a portable computer. Downton and Brooks (1984) and Dye, Newell and Arnott (1984) are examples of early efforts to use achieve such automated transcription.

SIZE OF ALPHABET

The larger the alphabet encoded on a keyset, the more difficult it is to use. A cognitive problem with such keysets is that you cannot hunt-and-peck. You must memorize the encodings, and this results in a longer training period, errors for infrequent chords, and re-learning problems for casual users. With the Microwriter, for example, it requires only about two to four hours to learn the 26 letters of the alphabet and the 10 digits. However, remembering all of the special symbols and punctuation is sufficiently difficult to discourage use of the device.

SEMANTIC LEVEL OF SYMBOLS IN ALPHABET

One reason that people have been attracted to chord keyboards is the prospect of improving the bandwidth of data entry by a human operator. But note that the rate at which an expert can "type" symbols on a chording keyboard is considerably slower than on a conventional typewriter. Devoe (1967) cites top rates of about 125 versus 800 strokes per minute respectively. Consequently, to increase bandwidth with chording, more information must be transmitted per stroke. We see this in Seibel's "Rapid-Type" which improved effective typing speed because common words were able to be abbreviated as a single chorded symbol. Likewise English Braille has three levels of encoding, each of which embeds more information into the cells. What we discussed earlier in the chapter was Grade 1 Braille, where a cells represented individual characters. With Grade 2 Braille, cells can represent standard abbreviations and contractions. Finally, with Grade 3 Braille, users can create their own abbreviations, etc., thereby creating their own personal shorthand.[1]

Similar information "packing" can be used in input techniques other than chording keyboards. Programmable function keys are one example. In this case, the issue is the trade-off between number of keys and the number of symbols (words or characters). With a non-chording keyboard, n keys gives us access to n symbols, compared to 2^n using a chording version. The question of "semantic load" is also very relevant in designing input systems for people with motor disabilities. Here, due to the relatively high overhead of each action, the issue is to get as much information out of each one. Demasco and McCoy (1992), for example, give a good discussion of word-based virtual keyboards and sentence compansion as techniques to obtain maximum bandwidth per user action. In reading this work, it is important to realize that these techniques have potential application for the wider computer user population, and warrent investigation. Finally, we the topic of semantic load will reappear when we discuss marking interfaces in a later chapter. Here the issue is whether the mark being recognized represents a character, word or entire command (sentence).

Discoverability

Discoverability is the property of user interfaces to reveal to the user that it is there, how it is to be used, and what the current options are. For example, most menu systems are fairly

---

[1] This ability to extend the standard notation in order to create a personalized short-hand exists in other notational schemes besides Braille and Seibel's Rapid-Type. See, for example, Pittman Shorthand, discussed in Chapter 13: Marking Interfaces.

self-revealing, in that each menu item makes explicit one of the user's options, assuming a few basic conventions are understood. On the other hand, command-line interfaces such as the Unix Shell, or MS-DOS, reveal little, if anything, to the user about what they can do next.

At the device level, the QWERTY keyboard is fairly self-revealing, since the labels on the top of each of the keycaps indicate what will happen if that key is pushed. However, even here there are problems, as illustrated by special characters that do not appear on a keycap. For example, most keyboards are not self-revealing when it comes to entering the "≠" character.

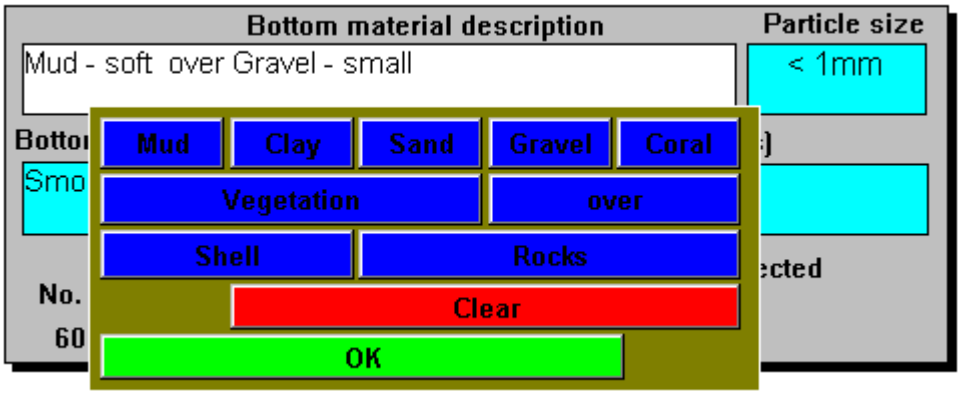

Figure 19 : A Visual Prompt for Data Entry with a Chord Keyboard (Australian Institute of Marine Science)

Since there is not a simple keycap-character mapping with chord keyboards, it is difficult to reveal to the novice how to enter specific characters, short of going back to the documentation. Hence the claim that in order to type on most chord keyboards, one must be able to touch type on them.

Whereas this is generally Science for the case, there are exceptions. Figure 19illustrates a technique developed by the Australian Institute of Marine Science which addresses this problem for certain cases. What they do is display a menu on the screen which illustrates the effect of different chords. In the example, each option corresponds to one or more fingers (thumb at the left) on a 5 button hand-held controller. By pressing the thumb and first finger, for example, they can activate "Shell" as their next entry, similarly "Clear" would be all but the thumb, and "Mud" just the thumb.

The value of the technique is that it provides prompts for the various chord combinations. The weakness, however, is that it limits the chords that can be used, since only chords of contiguous keys can be conveniently labeled using this approach.

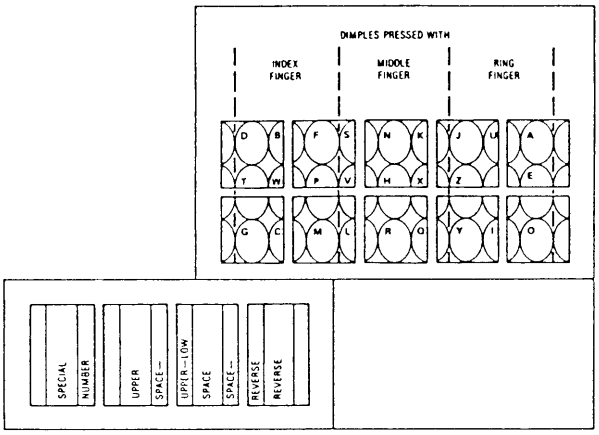

Figure 20: Encoding Characters and Words. (Rochester et al.,1978)

Figure 20 gives an example of how both characters and words are encoded on the keyboard developed by Rochester *et al.*, 1978).

ENCODING SCHEME AND HANDEDNESS

The encoding used is critical in minimizing the problems of learning, retention, and operation. For one-handed keysets, it also has an effect on handedness.

Not all people will use a one-handed keyboard in the same hand. The obvious example is with left-handed and right-handed people. Even with a single individual, however, the device may need to be operated in either hand. When just entering text (with a Microwriter, for example), the major hand is typically used. However, when entering text while also using a mouse (as in the Engelbart and English setup), the text is often entered with the minor hand and the mouse manipulated with the major hand.

We have already seen how the design can facilitate, or impede, the ability to physically use the device in either hand. With the Engelbart and English keyset, transfer is easy because of the symmetrical piano-like design. However, with the Writehander and Microwriter a separate physical device - designed especially for the other hand - must be used.

Hand-to-hand transfer presents even greater problems. At issue is how the codes memorized on one hand transfer to the other. Will the encoding on one hand be the "mirror image" of the other, or will spatial congruence be maintained? If the keyboard has a vertical orientation, the two will be the same, and the issue disappears. However, this is not the case with any of the one-handed keyboards discussed.

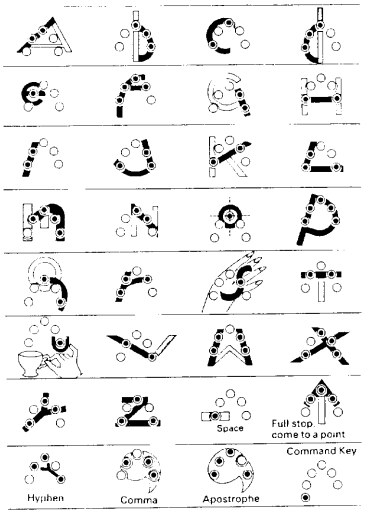

Figure 21: Encoding Scheme for Microwriter. (Microwriter Ltd.)

We can gain some insights about the hand-to-hand mapping by looking at errors in conventional two-handed typing. Figure 22 presents data from Munhall and Ostry (1983). If you compare the spatial congruence and the mirror-image pairs, you see that mirror-image substitutions occur much more frequently (in some cases, as much as 10 times more often than spatial congruence substitutions). This high frequency of mirror image substitution errors suggests that this mapping will be most likely when transferring the operation of a one-handed keyboard from one hand to the other.

TO DO: Insert Figure 10.1 from Munhall and Ostry

Figure 22: Errors in 2 Handed Typing (from Munhall and Ostry, 1983).

In contrast, a study of Gopher and Koenig (1983) suggests that the codes for one-handed chording keyboards will most naturally transfer by spatial congruence. If we examine the Microwriter, however, we will see how the question is largely a result of the encoding scheme used. In learning the codes for the Microwriter, mnemonic aids are introduced. In fact, at least three different types of mnemonic are used:

kinesthetic based: for example, the most agile (index) finger is used for the most common character, 'E' (similarly, thumb is used for second most common character, ' ').

word association: for example, "S" is typed using the "signet" ring finger.

spatial mnemonics: for example, 'L' is typed by pushing three keys spatially corresponding to the three vertices of the graph of the character. The encoding for 'J' uses the same scheme.

The encoding scheme for the full alphabet is shown in Figure 21. We will not deal here with the questionable practice of mixing mnemonic types (if you can't remember a character, you must first remember how it was encoded). What is important to note is that (in contrast to the findings of Gopher and Koenig, 1983) the first two encoding schemes will always transfer from hand-to-hand in a mirror image.

Conversely, the third scheme will nearly always maintain spatial congruence in transfer. (Otherwise, the mnemonic for 'J' would be shaped like an 'L' when we transfer hands.) Thus, we see that we can choose our encoding scheme and training to encourage one form of transfer or the other. Furthermore, if we mix the two types, as did the Microwriter, we severely impede the operator's ability to transfer the skill from hand-to-hand.

The implications of these issues go beyond the obvious. With longer alphabets, it could be argued that spatial congruence should be preferred (Gopher & Koenig, 1983). However, many mice can and should be considered chording keysets. But they generally only have 2 or 3 buttons and few commands. In this case, there is a strong argument to use mirror image encoding and have the most common (select) function triggered by the strongest (index) finger.

These are subtle but important issues that must be dealt with. Consider, for example, a 3-button mouse used with mirror image encoding, as suggested. Typically, users (left and right handed) use the button lying under the index finger for selection. Due to the asymetry of the hand, however, these would be the right and left buttons, respectively. To respond correctly, the system must interpret input according to which hand the mouse is in, and references in the documentation should refer to the finger rather than the button of the device.

But all of this can get us into trouble. Let us accept that a fundamental characteristic of good design is self consistency. We just argued for a mouse implementation based on mirror-image hand-to-hand transfer. But what happens when this mouse is used in combination with a chording keyset? If the keyset transfers encoding by spatial congruence, do we have an inconsistency that will affect the quality of the user interface? Yes, if at different times both devices must be used by both hands for a single user. No, if only one of the two devices transfers from hand-to-hand. Both cases can occur. Note that the issue of mirror image argued for in the case of the mouse was as much to improve the ergonomics for left-handed users as to enable one user to operate the mouse in either hand. With the chord keyset, the issue was hand-to-hand transfer for a single operator.

INTERFERENCE

All other things being equal, it should be no surprise that performance using a chord keyboard can be affected by other tasks. One such case is in two handed input where one hand is typing on the chord keyboard, and the other performing spatial tasks with a mouse. In this type of dual task, coordination of the two hands can be difficult, and require significant training. Therefore, task assignment and difficulty must be carefully considered and tested.

Another type of interference occurs when the typing and spatial tasks are performed by the same hand. This is seen, for example, when chording with the mouse buttons. Here, trying to type while simultaneously positioning the mouse will generally degrade the performance of both tasks. Typing even when the mouse is stationary may cause problems. Generally, the buttons of mice and tablet pucks are mounted so that their action is as close to vertical as possible. This is to keep the direction of button motion orthogonal to the plane of motion in positioning. However, buttons so positioned are generally not in the most comfortable location for chording. Hence, the DePraz mouse, which is optimized for chording, is held much like the Writehander, so that the finger tips rest comfortably over the keys. The penalty for this, however, is that the direction of force in button pushes is parallel to the plane of motion in positioning. Furthermore, in this position the mouse is held in a "power grip" (much the way you hold a ball). As a result, mouse positioning must be carried out using the larger muscle groups of the arm, rather than the wrist and fingers, which offer finer control.

One consequence of this discussion is a realization that a factor in choice of mouse or tablet puck should be whether the keys are to be used for chording or not, and what type of positioning is demanded.

CONCLUSIONS

Based on the literature and personal experience, I believe that chording keyboards have an important role to play in human-computer interaction. As Norman and Fisher (1982) point out, major improvements in methods for keyed input will only be achieved through a radical change from current practice. This is true for both novices and experts. The use of chord keyboards as an alternative means for keyed input is still under developed. This is, I feel, due to the range and complexity of the issues affecting their performance. To change this situation, research must investigate appropriate applications as much as technical issues. In my mind the issue is not if chord keyboards can be effective, but where and how?

TO DO STILL:

add discussion about:

data hand. Knight, L. & Rettner, D. (1989).

Lyons, K, Starner, T., Plaisted, D., Fusia, J., Lyons, A, Drew, A., Looney, E.W. (2004). Twiddler Typing: One-Handed Chording Text Entry for Mobile Phones. Proceedings of CHI'04, 671-678.

Lyons, K., Starner, T. & Plaisted, D. (2004). Expert Typing Using the Twiddler One Handed Chord Keyboard. *IEEE International Symposium on Wearable Computers*, 94-101.

Discuss trade-offs with harmonic vs arpeggiated chord keyboards. For info, see:
http://turton.co.za/chordingKeyboard/chordingKeyboard.html
http://trevors-trinkets.blogspot.com/2007/07/five-finger-keyboards.html

Add figure 14

Engelbart explains chord input: http://www.dougengelbart.org/pubs/augment-14851.html

	MOUSE	0 0 0	0 X 0	X 0 0	X X 0
	CHORD	**CASE 0**	**CASE 1**	**CASE 2**	**CASE 3**
1	0 0 0 0 X	a	A	!	show one level less
2	0 0 0 X 0	b	B	“	show one level deeper
3	0 0 0 X X	c	C	#	show all levels
4	0 0 X 0 0	d	D	$	show top level only
5	0 0 X 0 X	e	E	%	current statement level
6	0 0 X X 0	f	F	&	recreate display
7	0 0 X X X	g	G	‘	branch show only
8	0 X 0 0 0	h	H	(	g off
9	0 X 0 0 X	i	I	)	show content passed
10	0 X 0 X 0	j	J	@	i or k off
11	0 X 0 X X	k	K	+	show content failed
12	0 X X 0 0	l	L	-	show plex only
13	0 X X 0 X	m	M	*	show statement numbers
14	0 X X X 0	n	N	/	hide statement numbers
15	0 X X X X	o	O	^	frozen statement windows
16	X 0 0 0 0	p	P	0	frozen statement off
17	X 0 0 0 X	Q	Q	1	show one line less
18	X 0 0 X 0	r	R	2	show one line more
19	X 0 0 X X	s	S	3	show all lines
20	X 0 X 0 0	t	T	4	first lines only
21	X 0 X 0 X	u	U	5	normal refresh display
22	X 0 X X 0	v	V	6	Inhibit refresh display
23	X 0 X X X	w	W	7	All lines, all levels
24	X X 0 0 0	x	X	8	One line, one level
25	X X 0 0 X	y	Y	9	Blank lines on
26	X X 0 X 0	z	Z	=	Blank lines off
27	X X 0 X X	,	<	[	
28	X X X 0 0	.	>	]	
29	X X X 0 X	;	:	_	
30	X X X X 0	?	\	ALT	centredot
31	X X X X X	SP	TAB	CR	

Source: http://www.dougengelbart.org/pubs/augment-14851.html