# Chapter 13:

# MARKING INTERFACES

## Introduction

In this Chapter we discuss what we refer to as *marking interfaces.* These include what are often referred to as *paper-like* and *pen-based* interfaces. They also include a number of interfaces that have been called *gestural*. For the purposes of this book, what we mean by “marking interfaces” are interfaces where the primary mode of interaction involves the user entering marks into a computer, often much like the marks that one might make with a pencil or pen. This would include (but is not restricted to) systems where one enters text using handwriting, or annotates or edits a document using conventional proof-reader’s symbols.

With the growing use of pen-based computers, such as the Apple *Newton* shown in Figure 1, this class of interface is of increasing importance.

While the term “paper-like” aptly characterizes a number of interfaces falling into this category, we will not use the term for two reasons. First, as will be seen, some of the manifestations of marking interfaces bear very little resemblance to things that we do with paper. Secondly, there are styles of interfaces, such as form-filling using a standard GUI, which are paper-like, yet do not employ marking.

Likewise, we have chosen not to employ the term “pen-based.” While a stylus is often the preferred way to enter marks, it is certainly not the only one, so the term “pen-based” is overly restrictive. Fingers, mice and pucks can and have been used successfully for many of the interactive techniques that we will discuss.

Finally, people often refer to some of the styles of interaction discussed in this chapter as “gestural.” Certainly it normally takes a gesture to create a mark. Despite this, what distinguishes the interfaces discussed in this chapter is the mark that results from the gesture, not the gesture itself. It is the mark, not the gesture, is the basis for the interaction. Hence, we distinguish marking interfaces from *gesture-based* interaction, which is discussed in the next chapter. While this is not a distinction that is universally made, hopefully it help clarify our discussion of user interfaces.

It is significant that the word “recognition” does not appear in the chapter title. Recognition is often not essential for the success of marking-based interaction. The annotation of documents is one such example. Character recognition is a “black hole” which has sidelined many a product that could have otherwise been successful. One seldom hears someone saying, “I wish this piece of paper could understand what’s written on it!” However, one frequently hears, “I wish that I had that document with me," or, "I wish that I could find that document." Recognition is useful, but not always essential.

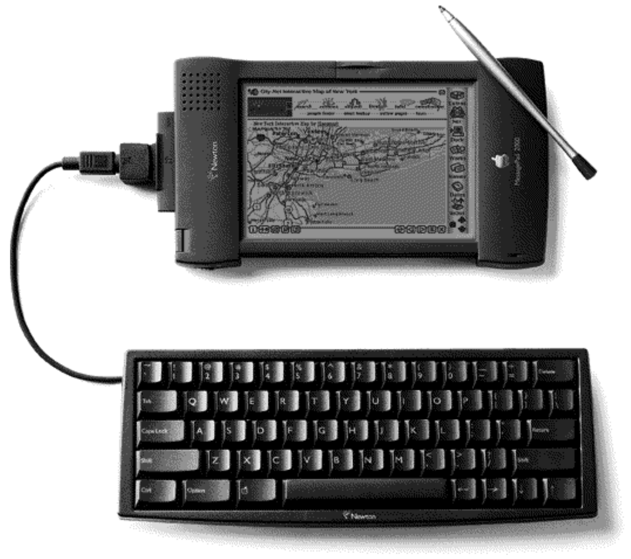


**Figure 1:** The Apple Newton MessagePad 2000 Computer.

*Stylus-controlled notebook computers are emerging as an important new class and style of computer. The Newton, shown above, is one example. However, while the approach is important, there are a number of problems to be solved before this class of interaction reaches its full potential. The design of the MessagePad 2000, for example, had to accommodate both pen and keyboard based interaction.*

One reason that recognition is often not as important as one might think is that the value of marking often lies as much in human-human, as human-computer, communication. This can be seen in tasks such as "chalk-talk" type interaction typically seen in meetings using whiteboards, and with the annotation and signing of documents, for example.

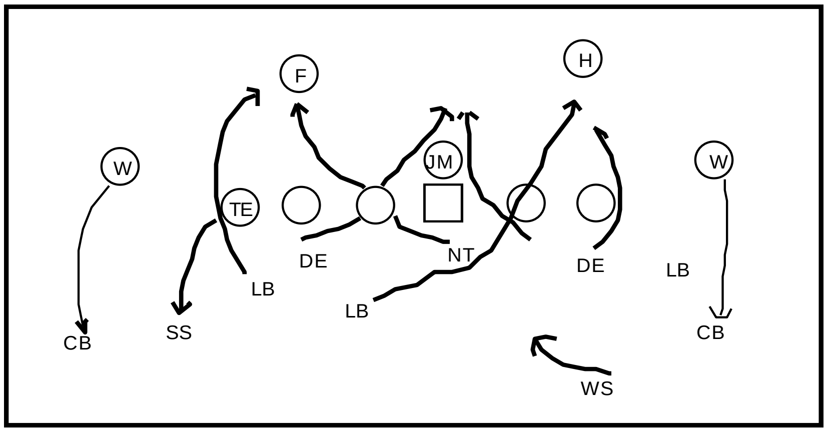


Figure 2: The 49er "Double Scrape."

*"Chalk-Talk" type marking notation is used to specify the spatial and temporal aspects of a football play.*

Consider the 49er "Double Scrape" example shown in Figure 2. This is a great example of how "chalk-talk" annotation can be used to notate very complex temporal and spatial information, such as a football play. The notation is something that every kid that has played sports can understand. It requires no computer training, and makes communicating this kind of information between humans easy and natural. In contrast, try specifying this same information to a computer using a conventional keyboard or standard GUI.

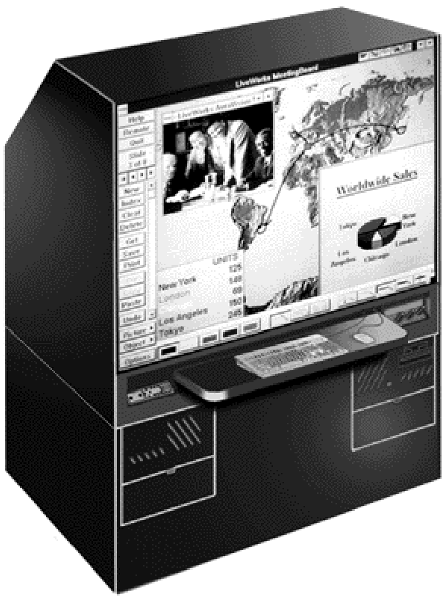


Figure 3:The Liveworks *Liveboard*

*The Liveboard is a 170 c.m. display with which one can interact with a wireless stylus or a conventional keyboard. It is an electronic whiteboard mainly intended to support meetings.*

With the introduction of electronic whiteboard technologies, such as the *Liveboard* illustrated in Figure 3, the need to support such chalk-talk type dialogues will grow, as will the expectation to be able to communicate with computers using this same type of fluid notation.

Already there has been some early work geared at enabling users to communicate the same type of complex spatial/temporal information to a computer. Perhaps the first example was a system called *Genesys* developed by Ron Baecker at MIT (1969). While this work on "picture-driven animation" was done early in the history of computer graphics, and not well known, it has significant relevance to future marking interfaces.

This is illustrated in the four frames of an animation from *Genesys* that are shown in Figure 4. Each frame shows a duck, as well as a dotted curve that goes from the lower right to the upper left of the frame. Hand-drawn digital ink lines make up both the curve and the duck. However, the duck and the curve serve different purposes. The duck is the protagonist of the animation. The curve defines the path along which this character is to move.

It also determines the speed at which the duck moves. This was defined by the speed, or dynamics, with which the curve was drawn by the animator. Each dot on this motion curve represents one uniform time interval. In each successive frame, the duck moves to the next dot. Where the dots are close together, the duck moves just a short distance from frame to frame. Conversely, where they are widely spaced, there is a correspondingly larger distance traveled between frames. Thus, the duck's speed varies inversely with the dot density.

Now all of this can be related back to our conversation about "chalk-talk" diagrams simply by imagining the duck being replaced by the squares and circles representing the football players in the 49er "Double Scrape", and each one having its own motion curve. Figure 2 can then be seen as a graphical script for an animation that we see in our mind's eye. The sad thing is that, despite the common usage of this type of diagram, and despite the existence proof of *Genesys*, today's computers are singularly unsuited for handling this kind of rapid, simple articulation of spatial/temporal relationships. This is even more disappointing, given how long ago *Genesys* was done.

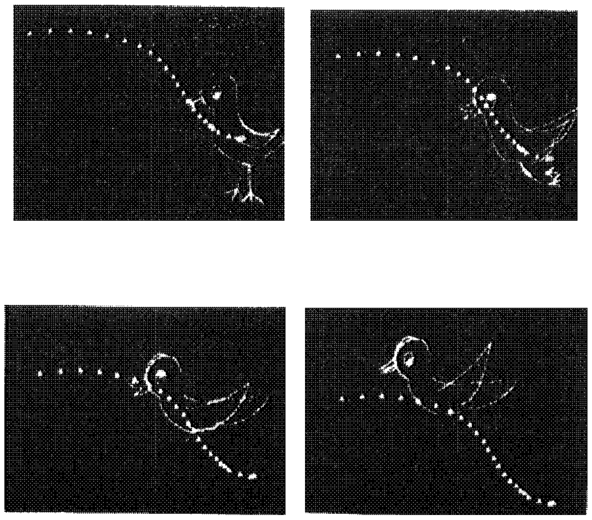


Figure 4: Picture-Driven Animation

*Four frames from an animation of a flying bird are shown. A line has been drawn to specify the path that a bird's flight should follow. The speed at which the line was drawn controls the speed of the flight. This is an excellent early and important example of the power of lines to specify complicated spatial and temporal information to a computer. (From Baecker, 1969.)*

**Comparison to other styles of interface**

To be done still:

- for comparative evaluation of marking and other input techniques: cover Gould & Salaun (1987); Compare Wolf (1988) to Wolf (1992) and incorporate differences (if any).
- for comparison of handwriting versus other interaction techniques cover Mahach (1989)
- for study on prognosis of pen-based interfaces, see Briggs et al, 1993.
- also, include discussion of Morrel-Samuels, P. (1990)

Wolf (1992) undertook three experiments to investigate the use of marking based interface techniques in working with a spreadsheet. In her first experiment, she compared three conditions. In the first condition, the spreadsheet (Lotus 1-2-3) was operate with in the conventional way, via a keyboard. The second and third conditions employed marking. In one, subjects marked directly on the screen. In the other, an indirect approach was used. The stylus was operated on an opaque tablet which was physically separate from the display. The main result from this experiment was to show that in both marking conditions subjects completed the tasks in 76% of the time taken with the keyboard interface. One interesting aspect of the results was that there was no statistically significant difference in performance between the direct and indirect marking conditions. (However, since a smaller and harder to read display was used in the direct condition, one should not immediately assume that these results generally hold. When using a measure that factors out some of the readability differences of the displays, the direct method was marginally faster )

| **Operation** | **Marking** (Expt. 1, 2, 3) | **Keyboard** (Expt. 1, 2) | **Keyboard/Mouse** (Expt. 3) |
|---|---|---|---|
| Mark Range | | Select using cursor keys, Return | Drag through range (or click for single cell) |
| Erase | | /re<br>Select range | Select range<br>Choose *Clear* from *Edit* menu<br>Click OK |
| Copy | | /c<br>Select range to copy<br>Move to target range<br>Select target range | Select range to copy<br>Choose *Copy* from *Edit* menu<br>Select target range<br>Choose *Paste* from *Edit* menu |
| Sum | | Type, e.g.,<br>" *@sum(A2..G2)* " | Select target cell<br>Type " *=sum(* "<br>Drag through range<br>Type " *)* " |
| Insert row (col)<br>(e.g., 2 rows) | | /wir (/wrc)<br>Select range | Select range<br>Choose *Insert* from *Edit* menu |
| Enter | Print | Type | Select cell<br>Type |
| Change | | F2<br>Change characters<br>(or retype cell) | Select cell<br>Drag through characters<br>Type |

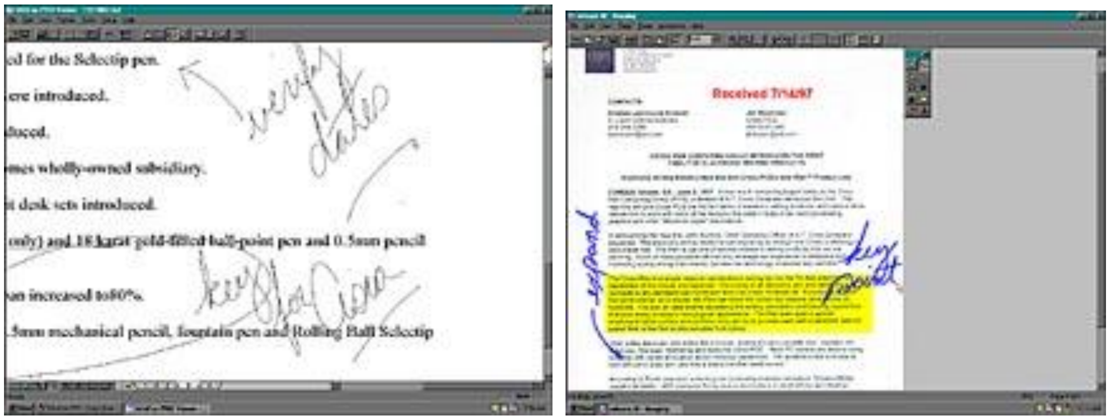
**Table 1:** Gestural, Keyboard and Mouse/Keyboard Commands used in the experiments of Wolf. (After Wolf, 1992)

The CRT display used in the keyboard condition in experiment 1 was also easier to read and larger than the LCD panel used in the direct marking condition. In order to factor out differences, a second experiment was run comparing the keyboard and direct marking conditions, using the same LCD display for each. The results were consistent with those of experiment 1, with the marking technique resulting in the task being performed in 69% of the time required or the keyboard technique.

In the previous two experiment, the non-marking spreadsheet was operated by keyboard only. There was a desire, therefore, to compare the marking technique to a spreadsheet that used a mouse as well as a keyboard. Hence, a third experiment was performed, this time comparing Microsoft *Excel* on an Apple Macintosh with the marking interface. On the average, subjects completed the tasks in 58% of the time required by those using the mouse/keyboard combination.

Given the fact that one would expect the mouse/keyboard approach to perform better than the keyboard alone, this result is surprising at first (since the marking time was only 76% and 69% of the keyboard times in experiments one and two, respectively, compared to 58% in experiment 3). However, this seeming discrepancy can likely be explained by the fact that the subjects in experiments 1 and 2 were experts in the keyboard technique, whereas there were two distinct populations (novices and experts) combined in the result for experiment 3.

This leads us to a very important observation that falls out of the results of this experiment. Throughout this book, we have been working from the basis that the objective of HCI is to accelerate the process whereby novices perform like experts. Well, this experiment's results give us another example of how this can be achieved through appropriate design.



ed for the Selectip pen.
ere introduced.
luced.
mes wholly-owned subsidiary.
t desk sets introduced.
only) and 18 karat gold-filled ball-point pen and 0.5mm pencil
an increased to 80%.
.5mm mechanical pencil, fountain pen and Rolling Ball Selectip

Received 7/14/97

**Figure 5**: Annotating of Electronic Documents

*The images highlight the figure/ground contrast between the handwritten annotations and the computer printed text in two electronic documents. The first is a document annotated using a stylus within CompuThink's Paperles Office , and the second is an electronic fax annotated within Symantec's WinFax PRO.*

Using the mouse/keyboard technique (which reflects state-of-the-art current practice), the novice users were much slower in task performance than the experts. Yet, using the marking technique, the performance of the experts was only marginally better than the novices, and both groups were much better than the experts in the mouse/keyboard condition. (Remember, that there were no "experts" in the marking condition, yet performance was significantly superior to the mouse/keyboard condition where there was expertise. Using markings, "experts" performed the tasks in 73% of the time taken in the mouse/keyboard condition, in which they actually had "expertise.") In summary, what we have is another example where, by virtue of the interaction technique, the novice is immediately enfranchised with the performance of an expert!

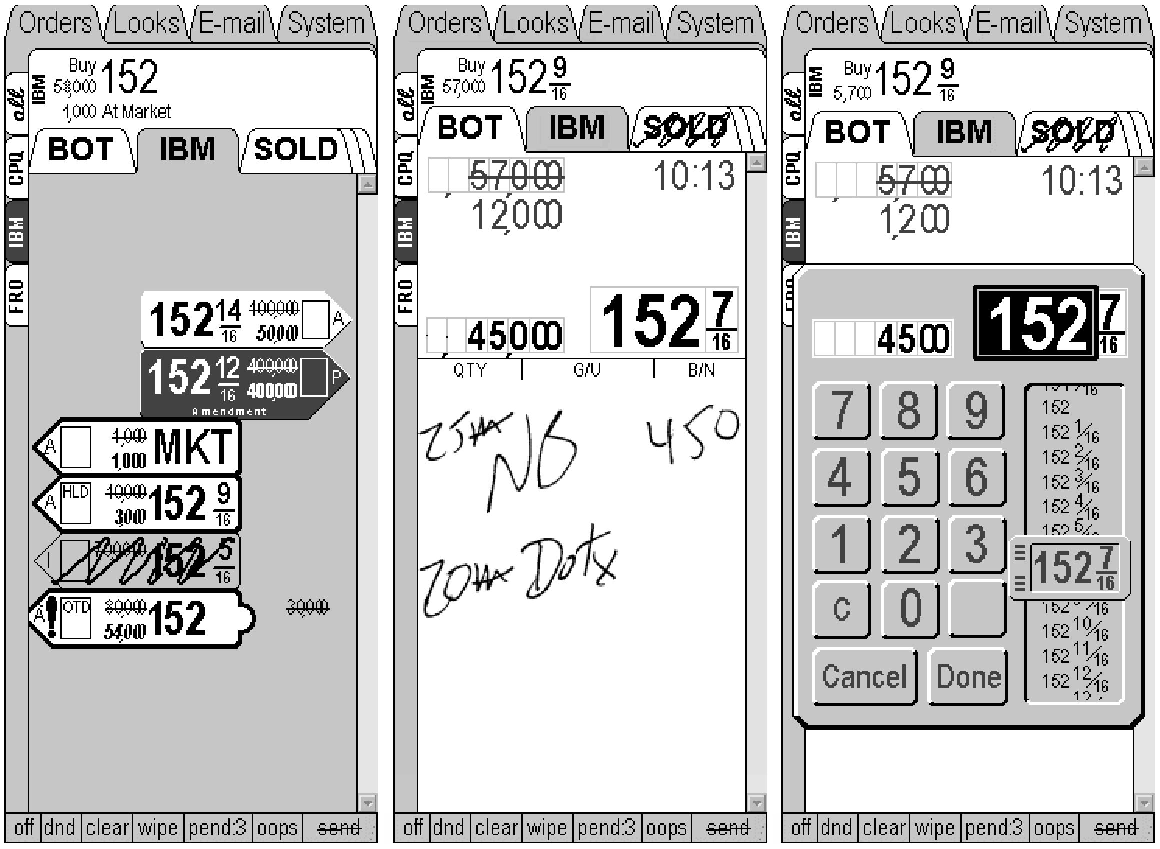


Figure 6: Hand Marking of Trading Form

*It is clear in the figure what was written by the computer versus by the trader. The contrast in line quality establishes a clear figure/ground relationship between the two. If there was character recognition, and what was written by the human was echoed in a pristine computer font, this interface would work less well. (Image: Brad Paley)*

### Marking *vs.* Recognizing

- paint programs
- Newman Marking, 1975 - ref: Newman (1986).

*Marking* was a program written by William Newman on the Xerox Alto computer. Marking was essentially a paint program like *MacPaint*, with one major difference: in Marking you could paint directly onto the page of an existing document, such as one containing graphics or text. There is a user's manual in the Alto User's Handbook (1976). Marking didn't do recognition of proof-reader's marks. Rather, like Wang's *Freestyle* system (discussed below), recognition was on the part of the user, not the machine: the hand-drawn marks on computer "typeset" documents established strong figure/ground relationships that provided natural and easy distinction between the annotation and the original document.

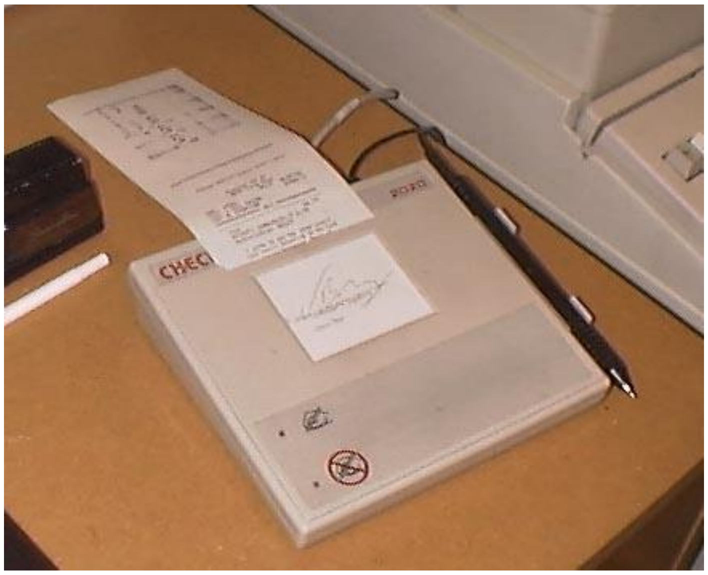


**Figure 7:** Pen Input in a Point of Sale Terminal

*Here a small digitizer is used to capture the customer's signature in a point-of-purchase credit card transaction.*

Emphasise the figure/ground issue
for annotation, see also Whittaker et al (1994), Weber (1994), Hardock (1991; 1993)
chalktalk=>black/whiteboard
MacPaint+projection = electronic whiteboard
coupled with telecommunications->shared drawing
CSCW
low level example, fax

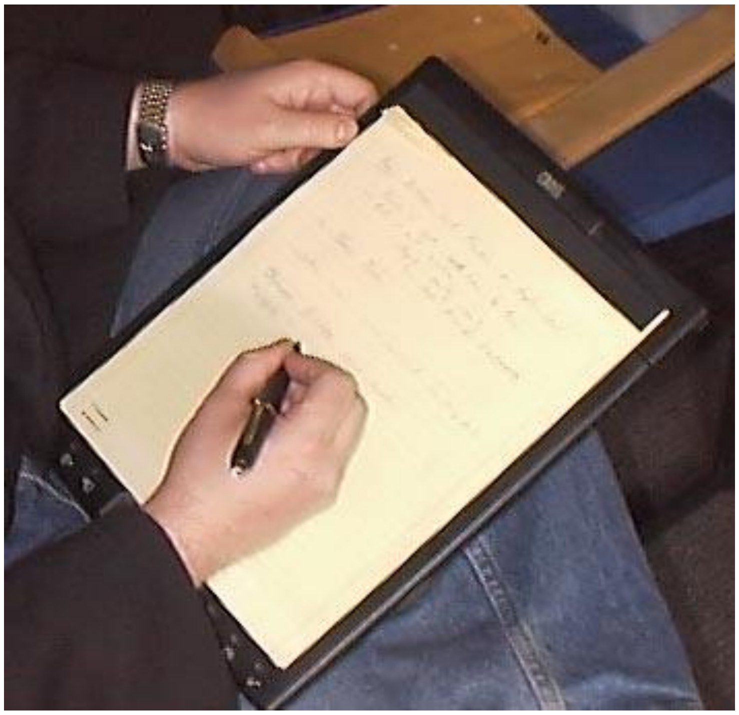


**Figure 8:** The CrossPad Portable Digital Notepad

*The CrossPad is an example of an emerging breed of embedded computers that are bringing us into the era of ubiquitous computing. It appears to be a conventional clipboard with a pad of paper. However, the contents of up to 50 pages of notes written on the paper can simultaneously be electronically captured in the clipboard. They can then be uploaded into a conventional PC for storage, indexing or optionally fed into a character recognition and converted into ASCII text.*

Wang *Freestyle* is an example of economical but effective design. It illustrates how much one can achieve with very little.

Freestyle is designed to support the annotation of documents. A usage model is, you send a copy of a document to someone by email, they annotate it using Freestyle, and then return it to you. You then make the changes. Annotations are made with a stylus, very much as would be made with paper and pencil. Voice annotations are also supported. Finally, while recording the voice, all stylus movement and action (pointing or marking) is captured as a form of animation, which is stored *synchronized with the voice.*

The copy of the document that is mailed is only a "dumb" bit-map (actually, tiff file) snap-shot of the original. The system has no recognition capability. The annotations are simply added to the "snap-shot." As with paper and pencil, this means that the user is unrestricted in the markings used, as long as they are comprehensible by the reader. But also like paper, the user then needs two versions of the document in order to make any changes: the annotated copy and the original.

The power of Freestyle comes largely from: (1) integration with the mail system, (2) the strength of the figure-ground contrast of hand annotations on top of "typeset" material. It is an example of careful design giving rise to a lot of benefit.

A potential problem with Freestyle is that it works so well that you may be led to have expectations which cannot be met. By not including recognition, Wang was able to side-step a hard problem, and get to market early. The challenge of this approach, however, is to avoid being boxed in by the design decisions made, and being able to make a smooth transition to a smarter system that includes recognition capability.

For more information on Freestyle, a demonstration appears on *SIGGRAPH Video Review 45* (1989). See also Perkins, Blatt, Workman and Ehrlich (1989), Francik and Akagi (1989) and Levine and Ehrlich (in press).

For a recent revisit to the concepts introduced by Freestyle, see RichReview (Yoon et al., 2014), which is a wonderful piece of work.

**Marking + Speech**

Wang *Freestyle*

- annotation with synchronized voice
- combination of marking & voicemail

HP's *FiloChat*

- portable device
- records voice as you take notes
- notes are index into speech
- see also Xerox work

RichReview (Yoon *et al*., 2014)

- updates the best of Freestyle and FiloChat
- couples with modern hardware and software
- gives promise to concept becoming broadly available and effective

From now on, unless obvious or otherwise stated, we shall be dealing with recognition systems.

**Recognition**

- early on-line character recognizers:
  - Teitelman (1964), Brown (1964), Bernstein (1964), Groner (1966), Ellis, T.O. & Sibley, W. (1967) Bernstein & Williams (1968) Ledeen (1967- [Described in Newman & Sproull 1973 (575-582) & 1979 (202-209)]). Suitable for student implementation. Description of Recognizer: Burr (1983).
  - Early math Anderson (email: anderson@rand.org, http://www.rand.org/pubs/authors/a/anderson_robert.html, http://www.rand.org/isg/staff.html,Telephone: (310) 393-0411 x7597
  - Berson, (1977). Also, Nakagawa(1990) presents survey of on-line character recognition for Japanese.
- Hardware for computer recognition of handwritten characters appeared quite early, being described as early as 1957 (Diamond, 1957)

Not recent, but valuable survey of automatic recognition of hand-printed characters with extensive bibliography is in Suen, Berthod & Mori (1980). A shorter, but more recent survey is Tappert, Suen & Wakahara (1988), which has an extensive bibliography. Discussion of evaluating different techniques is found in Litvin, 1982. Plamondon, Suen & Simner (1989) and Suen (1990) gives a good updates on the current state of the art in recognition techniques. One of the most important approaches to recognition (speech and mark) is the use of Hidden Markov Models. One example of this usage is NAg, Wong & Fallside, 1986. Rubine (1991a; 1991b) presents the first trainable recognizer for general markings. Mai, T. & Suen, C. (1990). present unconstrained recogniaer for numerals.

- character vs word recognition: eg., latter in Apple Newton. Provides context, so can be more accurate than isolated character recognition. Much the way that people can read words in contect that are misspelled. On the other hand, there is a real problem when dealing with words not in the dictionary. Then often strange results occur.
- Frankish, C., Hull, R. & Morgan, P. (1995). Interaction between recognition accuracy and user acceptance

Applications:

- BIOMOD: Groner, Clark, Berman, DeLand (1971),
- Ambit: Rovner & Henderson (1969)
- Electronic Paper: Brocklehurst, E.R. (1991)

Ambit-G was just another one of the many TX-2 graphical applications, most of which were driven with the TX-2 "Recognizer". (For example, a circuit layout program written by Fontaine Richardson et al was interesting enough to cause Fontaine to leave Lincoln Lab - the home of TX-2 - and start a little company called Applicon, an early leader in the field of CAD.) This was a symbol (one kind of the current large family of "gestures") trainer/recognizer front-end based on the Ladine symbol recognizer.

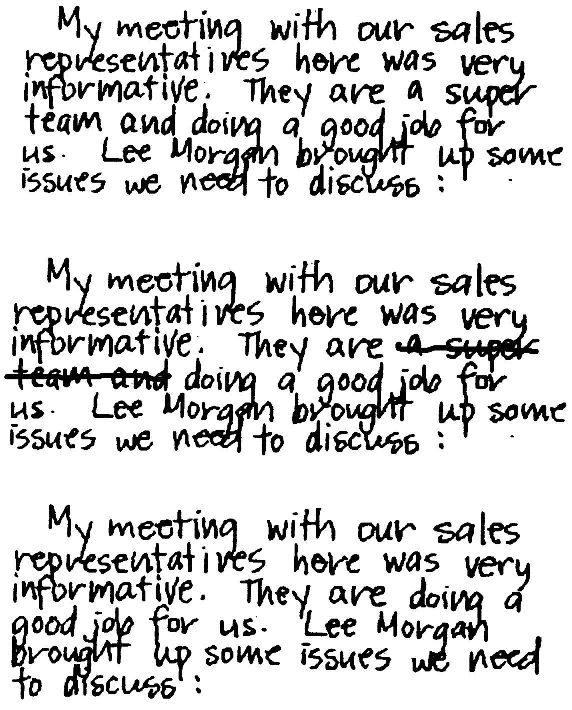
Application (1973) - several "gestures" design drafting system

- Irani, Wallace, & Jackson (1970) - for queuing systems. symbols in FW&C 206

• implications on computing environment. See Carr (1990) for description of design of operating system developed specifically to accommodate pen-driven interfaces (namely, the GO Technologies system).

### Aha!: A Digital Ink "Processor"

One of the best examples of the use of digital ink was a product called *Aha! InkWriter* from Aha! Software Corp. This was a "digital ink processor" in the same way that a product like *Microsoft Word* is a "word processor." With it, one can do things such as insert a word into a paragraph or add an item into a list. In each case, space is filled or made available, and things like word wrap behave as one would expect. An example of this is shown in Figure XX. Here, part of a sentence is deleted in a paragraph.



My meeting with our sales representatives here was very informative. They are a super team and doing a good job for us. Lee Morgan brought up some issues we need to discuss:

My meeting with our sales representatives here was very informative. They are ~~a super team and~~ doing a good job for us. Lee Morgan brought up some issues we need to discuss:

My meeting with our sales representatives here was very informative. They are doing a good job for us. Lee Morgan brought up some issues we need to discuss:

**Figure 9:** Automatic word-wrap with digital ink using Aha! InkWriter

*Like with a regular word processor, when text is deleted, the paragraph reformats itself and "fills in" the freed up space. With InkWriter, this is accomplished through an understanding of basic document morphology derived from the digital ink. First we see the original paragraph. Then we see the words "a super team" drawn through to specify that they should be deleted. Finally, we see the reformatted paragraph after the deletion.*

When one thinks of digital ink recognition, one usually thinks of the problems of recognizing the ink as a character ("a", "b", "c", …). *InkWriter* is interesting in that it uses the features of the ink to recognize document *features,* rather than *content.* It recognizes features such as words, paragraphs, lists and drawings - the *morphology* of the document - not the underlying content.

Certainly the characters constitute part of the document, and *InkWriter* does support character their recognition. This is something that one typically does later. The software implicitly assumes that during transcription, the speed of capturing the digital ink is most important, and that recognition all of the text, or key words, is something that happens during later revision.

*InkWriter* distinguishes between the digital ink of text *vs* drawings. As with text, the software is capable of recognizing basic morphological features, each of which can be edited and processed, including the recognition and "prettifying" of basic shapes. An example of this is shown in Figure XX.

This example shows how one can select hand drawn objects and treat them in a manner similar to what one would do with a conventional object-oriented drawing or CAD package. When selected, familiar "handles" are displayed that enable transformations such as scaling and translation to be performed.

In addition, selected objects can be "recognized" and transformed from hand-drawn digital ink to computer-generated geometry consistent with what was drawn.

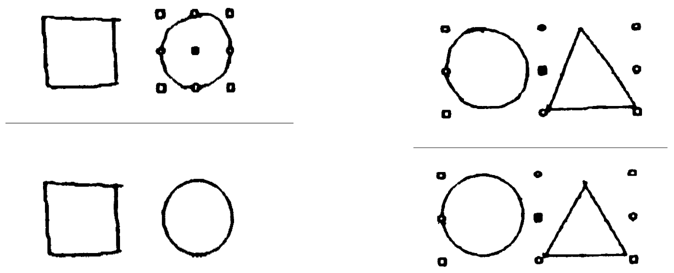


**Figure 10**: Processing of digital ink drawings with *InkWriter*

*As with conventional drawing programs, objects can be selected and manipulated. In the top left figure, a hand-drawn circle is selected and the editing "handles" are displayed. These enable the object to be scaled or moved. When selected, an object can also be "recognized" which is what is done here. The lower left part of the figure shows the hand drawn circle replaced by proper geometry. The right side of the figure shows the same thing, but this time for multiple objects.*

Perhaps include description of Moran's meeting support stuff too.
Poon, A., Weber, K. & Cass, T. (1995).
Weber & Poon (1994): Marquee

**Extending OCR to Lines**

• using pattern recognition and image processing techniques to convert hand-drawn figures and sketches into "finished" drawings. Cohen, 1982; Hosaka & Kimura, 1977,1982; Shirai, 1982;
• note, this is line recognition, rather than object or picture recognition
• consider part of issue on-line vs batch

**Semantic Power and Loading of Symbols**

• Like "Semantic loading" seen in discussion of chord keyboards.

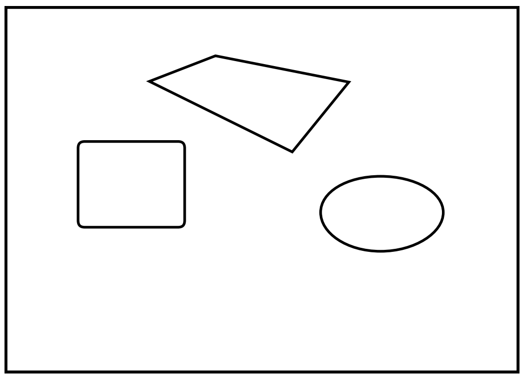
Simple Lexemes: Character Recognition

Augmented Lexemes: Embedded Meaning
• could include:
    • size
    • position
    • orientation
    • face (bold by pressure cue, for example)
    • style (italic by slope, for example)
• extreme case, signature recognition (Herbst & Liu, 1977)
    • may only recognize features, and not content (characters of name) at all!
• picture recognition: Herot (XX)

Symbols as Words, not Characters
• simple commands: delete
• music notation / GEdit (Kurtenbach & Buxton, 1991)

• circle for scope
• Pitman shorthand (Leedham, Downton, Brooks & Newell, 1984), Leedham, C.G. & Downton, A.C. (1986) Brooks & Newell (1985). Like Braille in terms of levels (see Chord chapter).
• simple drawing, Makkuni (1986)
• Lipscomb (1991)

Compound Symbols: from Words to Phrases
• proof-reader's symbols (give examples, include Kankaanpaa, 1988, ..., for example)
  • chalk talk
• domain specific operation, eg. Konneker, 1984.
• implies vehicle to support chunking & phrasing



More Chunking: Embedding Graphical Attributes
  • not just entering a shape
  • can include line type and face
  • can also easilly add other attributes, such as fill
  • Eller, Leyerle & Pardikar (1994)
  • note: fill not connected, therefore will be where problems lie
  • reliability will reduce with number of shapes in vocabulary
  • shapes in shapes?
  • "empty" fill for shape in filled bg.
  • nevertheless, contrast with traditional methods

**Text, Drawing and Page Orientation: Bimanual input applied to marking**
with paper, rotate page

eg guiard writing

- take stuff from article
- include text and image of animation
- relevance to pen input and "chunking" i.e., all in one step
- tie in with emaeging flat panels
- importance to simplified and more natural UI design

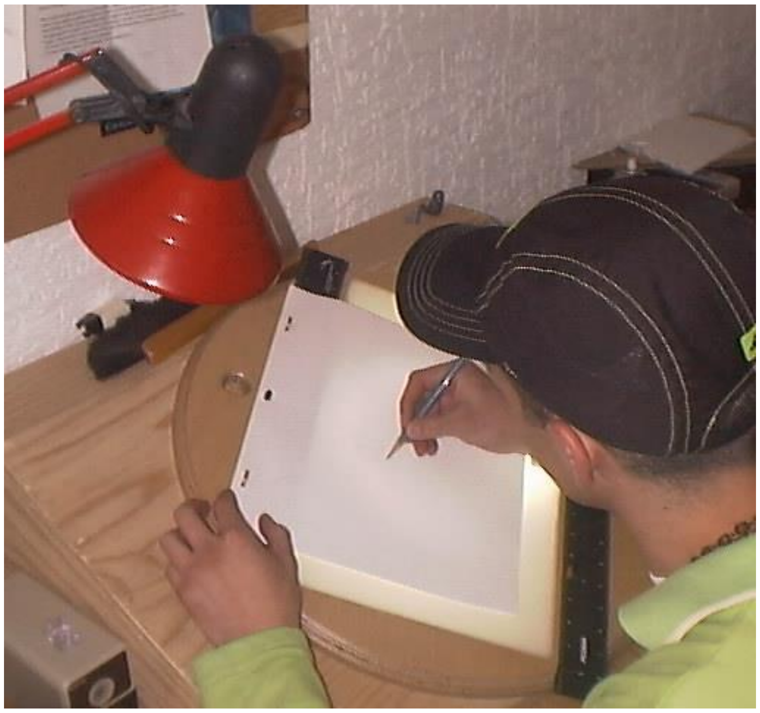


Figure 11:  A Traditional Animation Stand

*Note how the artwork is mounted on a rotatable base so as to maximize the ease of drawing at different angles on the page.*

Look at illustraions of text rotated

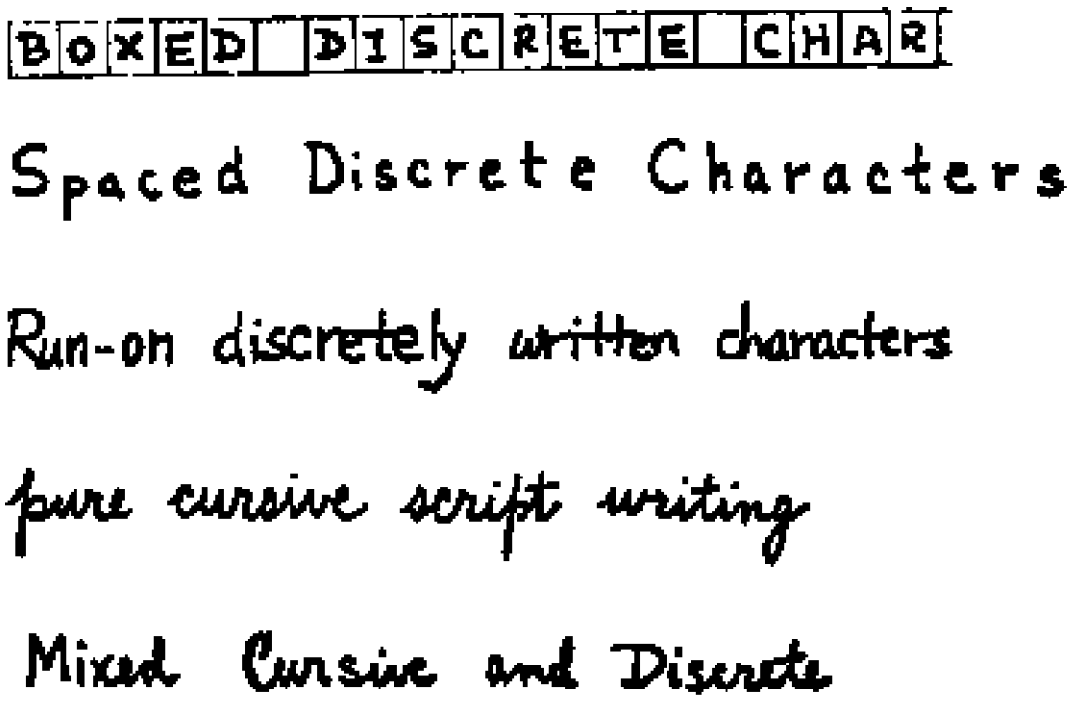

BOXED DISCRETE CHAR
Spaced Discrete Characters
Run-on discretely written characters
pure cursive script writing
Mixed Cursive and Discrete

Figure 12: Handwriting Styles.

*Different types of handwriting present different levels of difficulty in automatic recognition. The examples are presented in increasing order of difficulty. (From Tappert, 1984).*

**Variations in Script: Style and Penmanship**

Style:
• type of sign (for alpha-numerics: printed->hybrid->cursive)
cursive: Tappert (...) & Ehrich, Roger W. (1978)**, Ehrich, R.W. & Koehler, K.J. (1975)
• style => segmentation
• include Casio example

Penmanship
• quality of signal
• within and among users

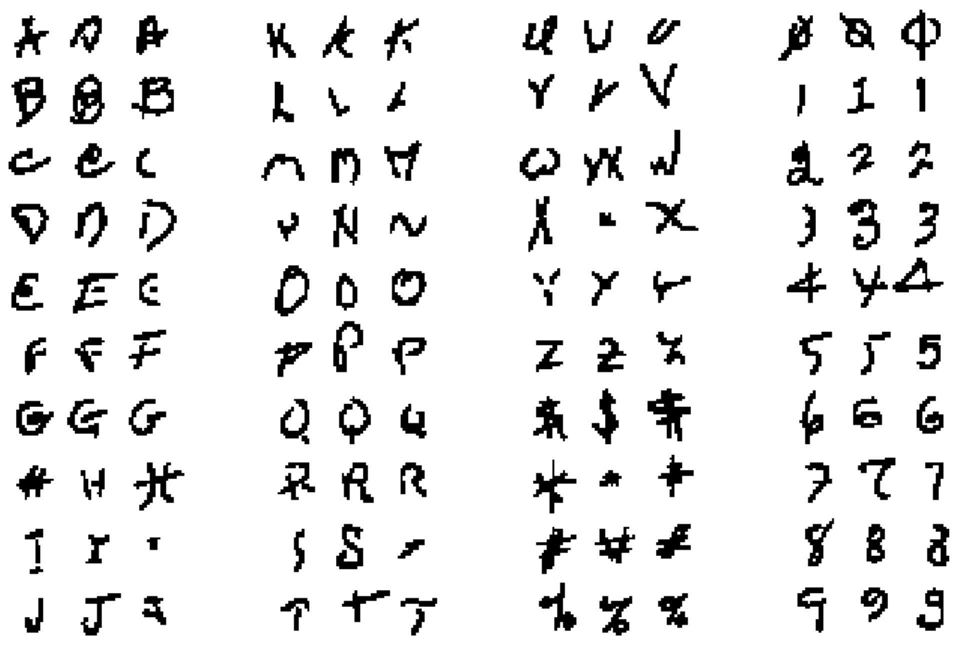


**Figure 13:** Variation of Printed Characters.

*Even when restricted to discrete character recognition, variability in penmanship makes the problem very difficult. The figure shows the range of characters that, nevertheless, can be recognized by a commercial system (from Ward & Blesser, 1985).*

**Speed and Efficiency**

• the "I can type faster than I can write, so who needs it?" syndrome
• first, clearly right for power text entry
• (at least until Pitman shorthand->text works)

• somewhere, include stuff on graphical keyboards, and T-Cube (Venolia 1994)

from semantic loading:
• shorthand can give function key-like power
• exploiting embedded information:
• Mathematical formulae: spatial layout has semantics. Implicit. Form & content compatible.
• annotation: may be able to type string faster,
• but what about direct translation from mind's eye to screen
• position, size, orientation typeface/style (bold/italics) implicit
• simple keystroke simulation quickly shows slowness of printing more than compensated for.
• home position on device (time multiplex device: mutual exclusive, and competition for hand resource)
• not back-and forth between keyboard and pointing device
• benefit maximized when editing characterized by relatively low volume of text entry and high volume of navigation (i.e., movement of entry point)
• coupled with 2 hands: home position on task (space multiplex devices, since no competition on hands as resource)
• eg, scrolling through document and selecting or editing words

• former example (1 handed), hand always on device, but would still have to move between text (for selection/editing tasks) and navigation aids (scroll gadgets)
• this case, 2 tasks partitioned between hands, each simultaneously in "home" position

Paper and Digital Ink: Bridging the Gap
include discussion of Anoto-type technology.
Especially include Adapx exploitation of printying glyphs, as with Excel (https://www.adapx.com/images/pdfs/Data%20Sheets/CapturxFormsforExcel_DataSheet_v1.0_w.pdf)

**The Graphics of Text**

We tend to forget when we think about text input that there is a graphical component to what we are doing. When we are just typing text on a keyboard using a word processor, we tend not to focus on this. Yet, when we are preparing slides for a presentation, or laying the text in on a graph, or doing text layout for an advertisement, this becomes far more “in your face.”

Take the text in the image below. The size, font, position, baseline and colour of the text, for example, is in many ways as important as what the text says.

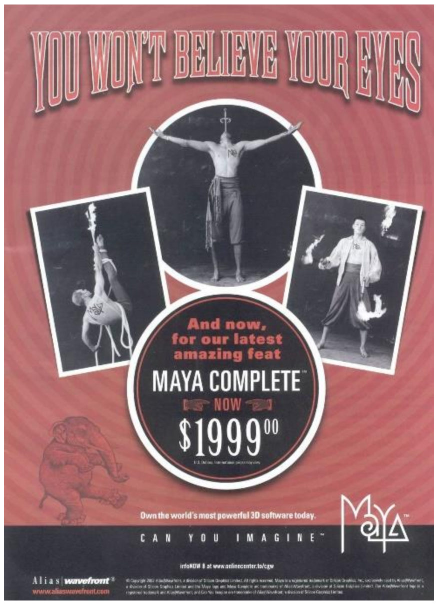

YOU WON'T BELIEVE YOUR EYES
And now, for our latest amazing feat
MAYA COMPLETE™
NOW
$1999⁰⁰
Own the world's most powerful 3D software today.
CAN YOU IMAGINE™
Maya™
infoNOW 8 at www.onlinecenter.to/cgw
Alias|wavefront®
www.aliaswavefront.com

Figure 14: Text as Graphics

In fact, as well as word processors, we have programs that are intended for doing text layout and graphic composition, as opposed to text composition and editing (as in traditional word processors.) What is often missed in thinking about text entry is how well suited marking style interfaces are for the former, and how nearly all thoughts about using marking to enter text is with regards to the former.

What is key to note in this regard is how, with a stylus, one can simultaneously input the characters of the text, the size, the position, orientation, base line, and even the face, all in one step. Furthermore, we can do so using skills that we have developed over the years in the everyday world, using a pencil.

Contrast this with the steps, and associated complexity, that would result if you did the same kind of layout with a conventional word processor, for example. And we don't have to be doing page layout or graphic arts for this to be the case.

Take the simple example of entering the following line of text:

$$\mathbf{2^2 \neq 22}$$

Consider entering it into your computer with the "fast" keyboard and the "slow" stlus. There are two problems that arise:

1. entering the superscript and
2. the "≠" symbol.

| Paper-like: | Conventional Word Processor: |
|---|---|
| • print "$2^2$" where I want it. | • type 22<br>• select second "2" by dragging through it (itself a fairly demanding task requiring fine motor control)<br>• select *Format* item on menu bar<br>• select *Character* from pull-down menu<br>• select *Superscript* from Character property sheet<br>• select the down arrow beside the "Size" field to reduce the size of the character by one point, from 10 to 9.<br>• select *OK* button to close Character property sheet<br>• select entry point for where to resume typing |

**Step 1:** Specifying the superscript

| Paper-like: | Conventional Word Processor: |
|---|---|
| • print "≠" character where I want it | • select  item on menu bar<br>• select *Keycaps* desk accessory from pull-down menu<br>• explore different combinations of "option" and "shift" until character found<br>• close Keycaps DA by selecting close box at left-top of the window<br>• type "≠" character |

**Step 2:** Specifying the ≠ symbol

Does this type of interface deserve the name "direct manipulation?"

---

- Speed of entry determined by graphical complexity of symbols (time to draw)
- Also is a function of the size & complexity of symbol set (can be cognitive load, and resulting delay, in remembering symbol in complex set.)

## Typing on a Graphical Keyboard vs Stroke Recognition

In pursuing pen-based interaction, our tendency seems to be to want to do with the electronic pen that which we do with pen and paper, namely, enter text by writing or printing. This tendency is natural, and reflects the concept of skill transfer that underlies much of what we have advocated in this book. It is, in fact, something that is too little practiced.

But here we have a potential counter example, or at least, an example of where blindly following any path might lead you to miss an opportunity. Consider the entry of basic text using a pen-based computer. In evaluating techniques, one that should be considered in addition to character recognition is the use of a graphical keyboard. As it turns out, as demonstrated by MacKenzie, Nonnecke, McQueen, Riddersma & Meltz (1994), tapping on a graphical keyboard can be both faster and more accurate than printed character recognition. (In their study, character recognition was 41% slower and had 8.1% errors rather than 1.1% for tap typing).

While not suggesting that character recognition has no value, we are suggesting that the appropriateness of typing by tapping with a pen should be at least considered in designing a pen-based application. This is something that can be determined on a case-by-case basis. There are no absolutes. But, of course, if the tapping approach is used, there are a number of ways to improve performance beyond the obvious.

One issue with graphical keyboards, regardless of whether they are operated by the finger, stylus, or a mouse, is that they consume valuable real-estate on the graphics display. This is of special concern on smaller hand-held systems. One approach to reducing this problem is to make the graphical keyboard smaller. As Fitt's law tells us, this comes at a reduction of typing speed. Nevertheless, we can make use of Fitt's law models to predict and evaluate the magnitude of this cost. Another alternative is to make the graphical keyboard semi-transparent so as to minimize the degree that it obscures the information on the display below it (Harrison, Ishii, Vicente & Buxton, 1995; Zhai, Buxton & Milgram, 1996). Finally, one could make the graphical keyboard "mobile" by controlling its position with the non-dominant hand while tapping characters with a pen or other device operate by the dominant hand. Coupled with semi-transparency, this would result in a specific instance of the *Toolglass* technique (Bier, Stone, Pier, Buxton & DeRose, 1993) discussed in the chapter on Two Handed Input.

A second thing which we can do in order to improve the speed of typing on a graphical keyboard is change the design of the graphical keyboard so as to better represent the physical properties of the hand. Hashimoto and Togasi (1995) introduced an oval design, shown in Figure 15, which better matches the range of reach of the hand when holding a stylus. They reported a 16% improvement in typing speed using this layout.

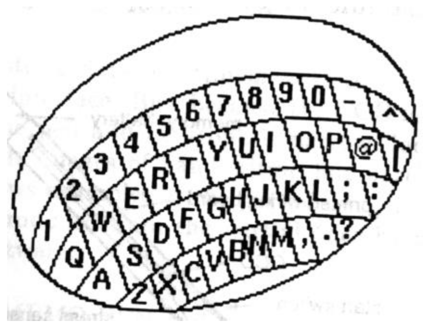


**Figure 15:** The Virtual Oval Keyboard (Hashimoto & Togasi, 1995).

*An improvement of about 16% in text entry speed was reported using this layout, which better reflects the range of reach of the stylus-holding hand.*

A third issue with entering text by tapping on a graphical keyboard is that it is still significantly slower than typing on a normal keyboard. On a graphical keyboard the optimal typing speed is mainly governed by Fitt's law, since you only type using one pointer (as opposed to eight fingers and two thumbs). For beginners, search time, rather than selection may dominate the task, but for those who know the keyboard layout intimately, Fitts rules the day. Thinking of things this way, one way to raise the typing speed is to reduce the number of keys selection tasks that need to be performed. One way to achieve this first practiced by Buxton and Kurtenbach (ref to come) and later by Hashimoto and Togasi (1995) is to combine tap typing with *marking menus* (Kurtenbach & Buxton, 1994). Using this approach, if one wanted just the simple character, one taps it as would be expected. However, if the key required a modifier, such as "Control" or "Alt", one makes a stroke starting on the graphical key cap, rather than a tap, where the direction (or posibbly shape) of the stroke indicates the particular modifier. Using this approach, one can also reduce the time needed to input other frequently used special characters, such as "Backspace", "Shift", "Delete", "Space" and "Return.

A good exercise which illustrates the value of some of the theory outlined in this book is to calculate the improvement that this should result in for various types of text, and to then test the accuracy of the calculations by experimentation. As a starting point, readers are referred to Soukoreff and MacKenzie (1995) for an analysis of the theoretical upper and lower bound for entering text on a graphical keyboard using a stylus. Of course, the exercise is not fully done, since Soukoreff and MacKenzie don't include the use of marking menus for special keys and modifiers in their analysis. As always, there is more to do.

Finally, there are other approaches to implementing graphical keyboards which are intended to optimize text entry using a finger or stylus. While not strictly *marking interfaces,* they seem best to fit into the current discussion.

The first of these is the *Fitaly One-Finger Keyboard*. The keyboard is laid out so that the most likely next character is adjacent, or near to, the last character entered. The idea is to achieve efficiency by reducing hand or finger movement. The keyboard layout is illustrated in Figure XX.

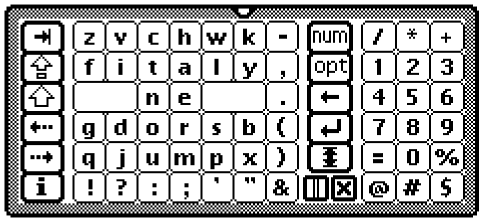


**Figure 16:** The Fitaly One-Finger Keyboard as it appears on the Newton MessagePad

If we think about this keyboard in terms of Fitts' law, we see that target size (*W*) is determined by the size of the graphics of the keyboard. Also, for a given keyboard size, it is mainly the distance, or amplitude (*A*) of movement which they are attempting to reduce. Clearly the reduction will be determined by how well the layout fits the vocabulary used. While the layout may help in English, it may make things worse in some other language (such as French or "C++"). Doing the analysis and contrasting the benefits against the cost of learning a new keyboard layout (remember, that unless the user knows the layout intimately, that there will be a visual search term in the analysis, $T_m$, in the terminology of the Keystroke Level Model.) MacKenzie, Zhang, and Soukoreff (1999) have actually performed the analysis and test comparing a number of graphical keyboard layouts, including the *Fitaly*, *QWERTY*, *Alphabetic*, *telephone keypad*, and something called the *JustType* layouts.

There are other approaches to keyboard layout that are intended to accelerate input. The last one that we will look at briefly is called *T9* (from "typing on nine keys") from Tagic Communications, Inc. While, the keypad is small, nevertheless there is only one keystroke per character. To get the whole character set on 9 keys, each key contains multiple letters (the "5" key, for example, can be used to type "J," "K," or "L,"). The T9 algorithm automatically determines from all the possible variations what word you are typing by matching your keystrokes with completed words in a linguistic database. Clearly, one of the issues here is that you need a different dictionary for whatever language you are using. The keyboard is shown in Figure XX.

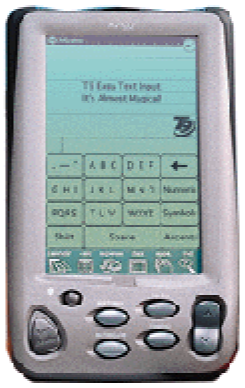


**Figure 17:** The T9 Graphical Keyboard on the Texas Instruments' Avigo PDA.

### Shorthand Marks

The use of shorthand versions of symbols is one way to increase the speed of entry in marking-based interfaces. This approach can be used for entering alphanumeric and graphic data. Figure 18, for example, gives an example of how a characiture of a transistor symbol can be used as a form of graphical shorthand. The characiture visually resembles the symbol that it represents (thereby providing a mneumonic aid), and is much easier and faster to draw. While not all symbols lend themselves to such simplified representation, it is worth the designer's time to explore the degree to which it is afforded.

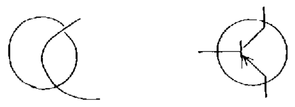


**Figure 18:** Using Simple Characiture to Specify Complex Object

*The choice of shorthand symbol need not be between representative drawing or some graphically unrelated symbol. This example shows how a simple characiture can capture the key properties of the intended symbol (a transistor), thereby providing the desired mnemonic value, while still being simple to draw and recognize. (From Newman and Sproull, 1979, p. 180.)*

73 ?

Shorthand can also be employed to facilitate the entry of alphanumeric data. One approach is to adapt traditional shorthand such as Pitman shorthand, to computer usage. (See Figure 19.) This has been investigated by some researchers, such as Leedham, Downton, Brooks, and Newell (1984); however, the approach is problematic. On the human side, there are relatively few people who are skilled in shorthand, and acquiring the skill requires considerable training. Technically, conventional shorthand also causes some serious problems due to the fact that the symbols correspond to phonemes, not alphanumeric characters. Thus, even if one can enter the shorthand, and it is correctly recognized, it must then be converted from phonemes to text. This is a rather difficult task that pushes the state of the art.

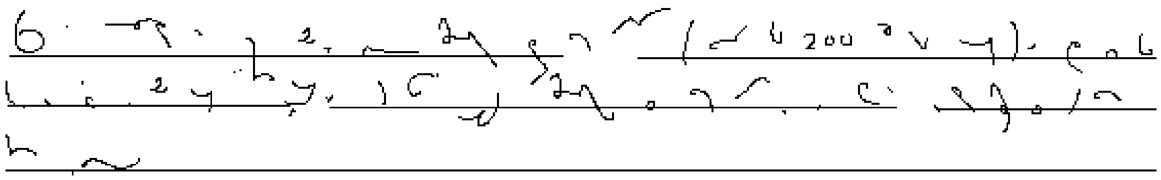


Figure 19: Pitman Shorthand

*The above example of Pitman Shorthand is a transcription of "This is an example of Pitman shorthand. You can transcribe speech very rapidly (greater than 200 w.p.m.). However, you then have to convert the shorthand into English. So, while the initial transcription is very rapid, the overall two-step process is much more tie consuming." (transcription courtesy of Elizabeth Russ)*

By having a one-to-one correspondence between shorthand and longhand symbols, the second translation phase seen with Pitman shorthand can be avoided, thereby rendering the use of shorthand more viable in many cases. One such example is the music shorthand (Buxton, Sniderman, Reeves, Patel & Baecker, 1979) discussed in Chapter 7, in the section on *Chunking and Phrasing*.

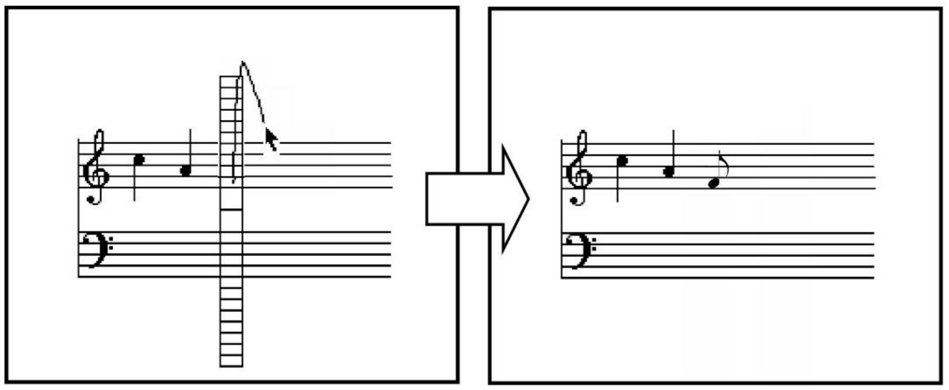


**Figure 20**: Entering an 8th Note Using Single-Stroke Shorthand

*As described in Chapter 7, the SSSP music editor enabled notes and rests to be entered with a mouse or stylus, using single-stroke shorthand. The shorthand was mnemonic in that the entry stroke related to the desired symbol. On the left side, an eight note F is specified to follow the quarter note A. The ladder helps guide the entry point, starting the stroke in the lowest space on the staff determines it is an F, and the shape – representing the stem and one flag – indicates that the duration is that of an 8th note. On completion, the rough hand-drawn stroke is replaced by a "typeset" version of the desired symbol. This is what UniStrokes and Graffiti did, but for alphanumeric rather than music symbols. As we shall see, the idea of using such a single-stroke shorthand precedes computers by a long-shot – going back to Roman times.*

One of the main attributes of the musical shorthand was that all of the attributes of a note (pitch, duration, entry point) could be specified in a single gesture. For example, on paper, one would typically write an 8th note in three steps:

- *Duration*: the note form (note head, stem and flag)
- *When*: the entry point along the horizontal “time line” of the music notation
- *Pitch*: where, vertically, the note appears on the musical staff

In the music shorthand discussed in Chapter 7, it can be entered as a connected up-down stroke. Hence, due to the economy of motion, a significant increase in transcription speed can be achieved over pen and paper. (Remember that, unlike paper, what is displayed does not have to resemble graphically the symbol written.) There is no need for training the recognizer for different users, since all users use the same simple shorthand symbols. And since the shorthand symbols are articulated in a single stroke, or a connected series of strokes, the recognizer is much simpler than conventional recognizers. We shall go into this in more detail below, in our discussion of segmentation.

Not surprisingly, the idea of a shorthand that uses efficient single stroke symbols has also been developed for entering alphanumeric characters. One such example is the *T-Cube* system developed by Venolia and Neiberg (1994). This system used strokes and radial menus as an alternative to character and handwriting recognition, or graphical keyboards for entering text with a stylus. The key to the interface was a circular graphical target which had a central “bullseye” surrounded by eight additional “cells”. Text was entered by making a selection from one of nine different pie menus, each of which was invoked by depressing the stylus in one of the cells of the target. This is illustrated in Figure 21.

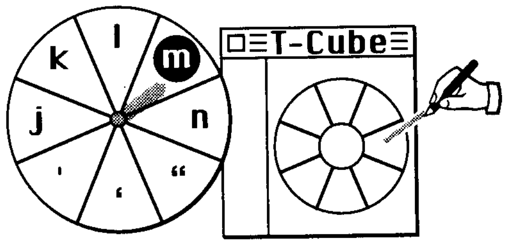


Figure 21: The T-Cube Target

Each of the nine pie menus contained eight characters from which one could choose. Figure XX, for example, shows selecting from the menu associated with the eastern-most cell of the target. The full set of menus is illustrated in Figure 22. It is worth noting that, just as with a traditional QWERTY keyboard, modifiers such as “shift” could be used to access more than 8x9=72 characters.

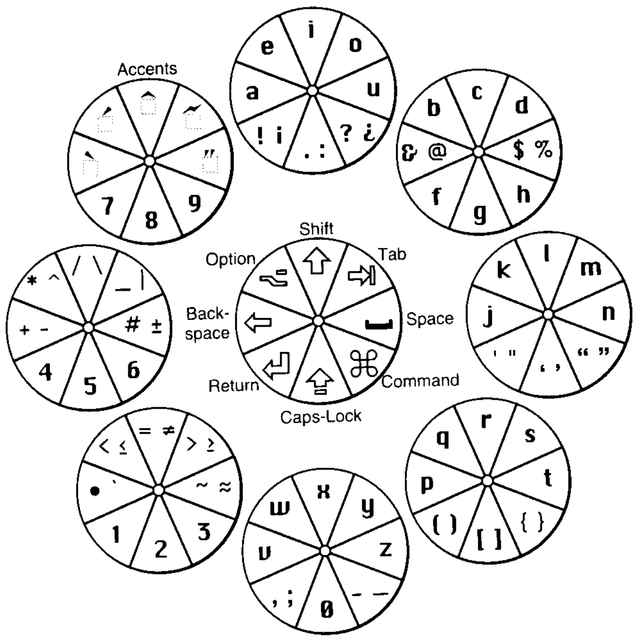


Figure 22: The T-Cube Menus

The T-Cube approach was never used in a commercial product. Its main drawback was the amount of time that it required in order to become proficient in it – to memorize the contents of all of the pie menus. One attribute that it did have, however, which distinguishes it from the other shorthand scripts that we will discuss, is that it was *self revealing.* That is, if you couldn't remember the stroke for a particular character, like with *marking menus,* you could find it by the graphical equivalent of "hunt-and-peck," namely, by exploring the menus.

Perhaps the first example using single stroke shorthand for entering *alphanumeric* characters into computers is the *Unistroke* shorthand of Goldberg and Richardson (1993, 1993 video). The basic Unistroke alphabet is shown in Figure 23.

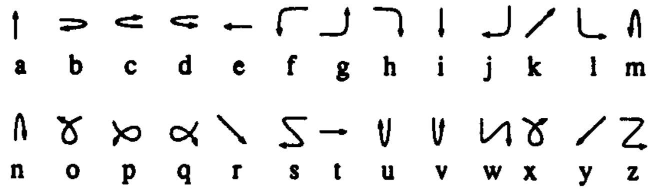


Figure 23: The Unistroke Alphabet

In designing the Unistroke symbols, Goldberg's prime consideration was speed of entry. Thus, the most frequently used characters are the fastest to draw. However, in the "there is no free lunch" department, this emphasis on articulation speed comes at the expense of learning speed. That is, in many cases (consider the symbol for "a", for example), any mnemonic value of the symbol is sacrificed in the name of efficiency of execution. This is a classic trade-off. Despite some efforts to facilitate the learning of Unistrokes, such as illustrated in Figure 24, they are still harder to learn than they might be; however, once the initial learning is accomplished, from then on one theoretically benefits from their efficiency. The assumption is that the long-term pay-off is well worth the higher, but one time, up-front investment.

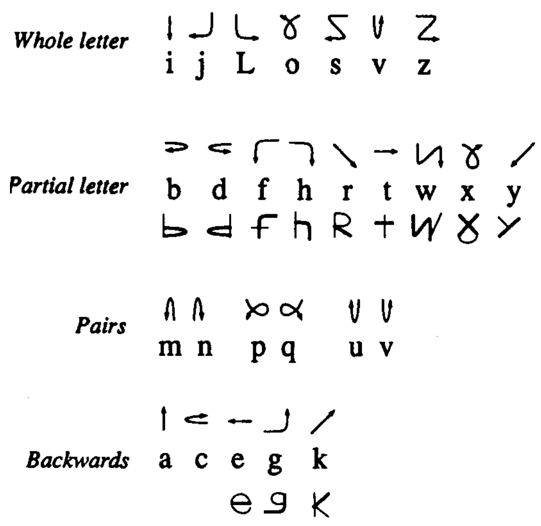


Figure 24: Unistroke Mnemonics

The risk with this approach, however, is that this initial investment is sufficiently high as to discourage adoption of the system in the first place. That this might be the case is suggested by the commercial success of derivative shorthand, *Graffiti,* (illustrate in Figure 25). Graffiti went the other way in the efficiency of learning *versus* articulation tradeoff. One can learn the basic alphabet in a matter of minutes, but this is at the expense of optimal speed of entry, when compared to Unistrokes. MacKenzie and Zhang (1997) provide a good discussion of how very intuitive Graffiti is to learn. They show that with only one minute of training, accuracy of about 86% is obtained, and after five minutes of practice, this goes up to 97% accuracy. Furthermore, retention after one week is still 97%.

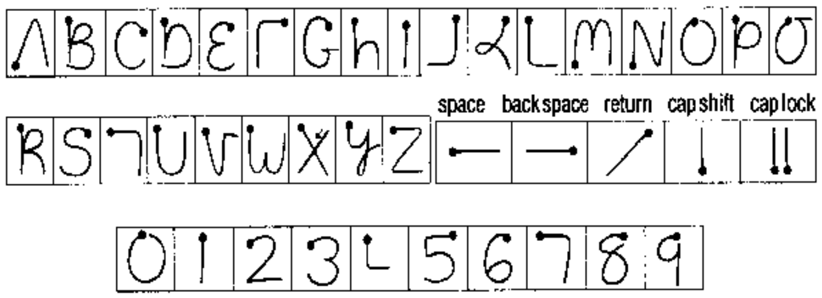


Figure 25: The Basic Graffiti alphabet

Finding the right balance in this learning *vs.* entry speed trade-off is extremely delicate. For example, the advantage of learning Graffiti over Unistrokes is largely restricted to the basic alphabet and digits (which, admittedly usually make up the majority of characters entered). When it comes to special characters, such as punctuation and special symbols (as illustrated in Figure 26), the difference in ease of learning or retention is probably negligible. To emphasize this point, the performance statistics given in MacKenzie and Zhang did not include special characters, so the performance measures that do exist are based on only a subset of the full alphabet. [**Note to self**: Incorporate discussion of Castellucci, S.J. & MacKenzie, I.S. (2008) here.]

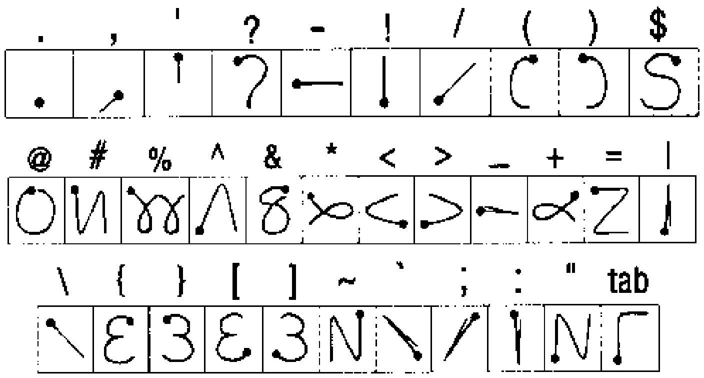


Figure 26: Graffiti:  Special Characters

*Contrast the intuitiveness of the special characters of Graffiti with that of the main alphanumeric characters seen in the previous figure.*

The extreme case on one side of the speed of learning *vs* speed of execution tradeoff is to simply use conventional printing or handwriting, and forget trying to accelerate input at all.  In so doing, of course, we leave the domain of shorthand altogether.  Close to this end of the spectrum is another single stroke system, *Allegro*, for the *Psion* Series 5 PDA from *Papyrus Software*, illustrated in Figure 27.  Here, one gets slower entry speed than *Unistrokes* or *Graffiti,* but can achieve faster entry than with conventional printing.  The system requires some initial learning, since the characters must be written in a particular way.  The real benefit of *Allegro* may well be improved recognition accuracy due to the constrained writing and single stroke alphabet, rather than improved entry speed.  However, there may be some benefit in entry speed, simply due to the time-motion efficiency of the single stroke alphabet. As is too often the case, there is no empirical data to help one make comparisons.

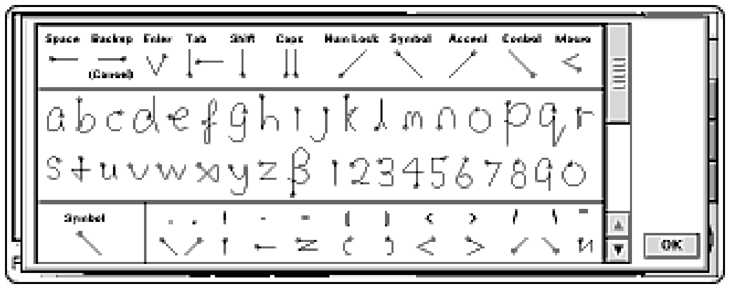


Figure 27:  Allegro Notation

*Allegro is a notation that uses a single stroke to enter each character.  Each character looks almost exactly the way it would in normal printing. However, in Allegro, characters are entered with a single connected stroke.  Because of the greater complexity of the strokes, compared to Graffiti or Unistrokes, Allegro will not be as fast to enter.  However, it is possibly faster to learn than Graffiti, but even this is not sure given the speed of learning Graffiti, and the fact that Allegro characters must be entered in a special way.  Taken Together, Unistrokes, Graffiti, Allegro, and the recognition of conventional printed characters provide the basis for an interesting study in the relative design trade-offs among learning time, text entry speed, and recognition accuracy.(courtesy of Papyrus Software)*

The final single stroke writing system that we will look at is *Quikwriting* (Perlin, 1988).  This is an interesting design variation on the previous examples in that it is to them what cursive writing is to printing.  It could be called a *cursive single stroke writing system.*  The claimed advantage of the approach comes mainly from the property that one need not lift the pen between characters, hence its cursive nature.

As is illustrated in Figure 28, one enters characters using Quickwriting in one of four modes: lower-case, upper-case, numeric, and punctuation, respectively. Notice that in each mode, the characters are grouped among eight different regions. It is important to note that there is always an odd number of characters in each of the eight regions.

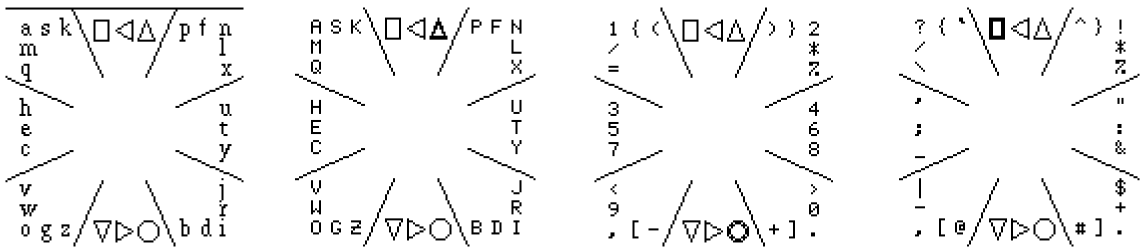


Figure 28: The Quickwriting Templates

*Quickwriting (Perlin, 1988) is a "cursive" single stroke writing system. In entering characters, one is in 1 of 4 modes: lower-case, upper-case, numeric, or punctuation, shown above. The characters in each mode are arranged in one of 8 regions, with an odd number of characters in each region.*

With Quickwriting, one always begins and ends the entry of characters from the same location: the middle of the template. It is from this property that the cursive nature of the system derives, since it means that the end position in writing character *n* is the start position in entering character *n+1.*

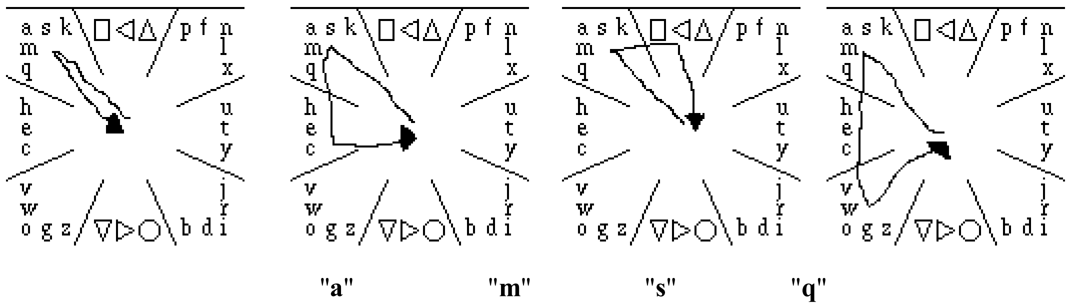
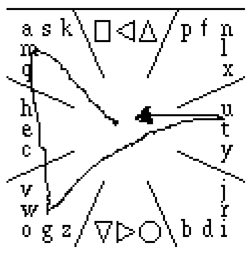


**Figure 29:** Entering Characters with Quickwriting

*To enter a character, one moves from the centre of the template into the region where the desired character resides. If the desired character is the centre character in the region, it is entered by returning to the starting point, as with "a". If the desired character is one step counter-clockwise from the centre character, one moves one region counter-clockwise before returning to the start, as with "m." If the desired character is one step clockwise from the centre character, one moves one region clockwise before returning to the start, as with "s." If the desired character is two steps counter-clockwise from the centre character, one moves two regions counter-clockwise before returning to the start, as with "q," and so on.*

One enters characters by moving from the central start region of the template into the outer regions and back. To select one of the characters in one of the eight regions, one moves from the start into that region. The path that one follows back to the start point is what determines which of the characters in that region is entered. Moving directly back to the start selects the centre character in the region. In general, one selects the rest by moving as many regions clockwise or counter-clockwise that the desired character is from the central character in the region. This is illustrated by examples in Figure 29.

The technique for entering a sequence of Quickwriting characters cursively is shown in Figure 30. Here one sees how the end of the entry of the "q" is continued to become the start of the entry of the "u."



"qu"

**Figure 30:** Cursive Entry of a Character Sequence

*Two single stroke characters can be entered in a cursive manner by having the end of one become the start of the next. In this case the characters "qu" are entered.*

All of these single stroke notations are interesting for at least two reasons beyond the potential for faster input. First, since each character is fully specified by a single stroke (or connected series of strokes), there is no *segmentation problem.* Second, single stroke entry systems afford what can be called *heads up writing.*

The segmentation problem can be seen in the example of entering a small circle followed by a vertical stroke. With a single stroke character set, the system need not invest any energy trying to figure out whether they are intended to signify “o l” or “d”. If a “d” was intended, the circle and vertical stroke would have been connected. Hence, unlike printing conventional characters, there is no need to print left to right, or within the confines of bounding boxes. Characters can even be recognized when the shorthand symbols are written one on top of the other. This is what happens, for example, using Graffiti on the *PalmPilot* electronic organizer, illustrated in Figure 31, for example. With this device, shorthand symbols are entered one on top of the other at the bottom part of the screen, and the resulting characters appear in conventional left-to-right order in the main upper portion.

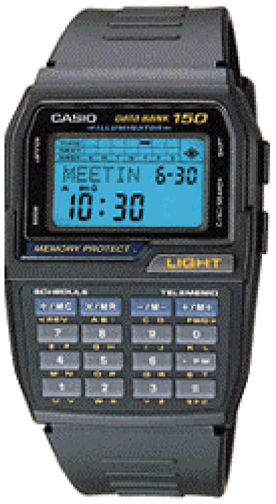
Figure 31: The US Robotics PalmPilot

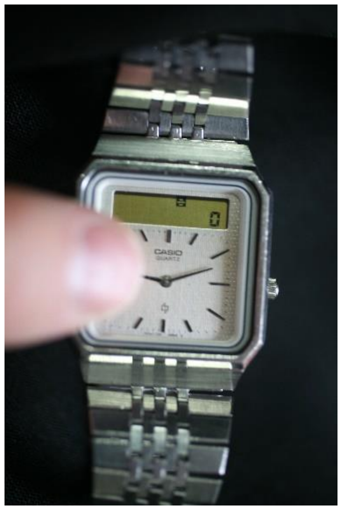
The second attribute that makes single stroke notations interesting is that they permit *heads up writing.* Because of the lack of segmentation problems, devices can be designed such that one need not look at the “page,” or display, when entering data. Thus, unlike paper and pen technologies, one can visually attend to the whiteboard in a lecture, or the document which one is transcribing, and still take legible notes. So, with careful design, we can achieve the handwriting equivalent of touch-typing.

A design situation where this might be useful is in entering alphanumeric data into a wristwatch based electronic organizer. Today, text entry to such devices is done using a micro-keyboard attached to the watch, such as that illustrated in Figure XX. However, if the face of the watch is touch sensitive, then one could employ a finger to enter data, using a single stroke notation like Graffiti. The second watch illustrated in Figure XX is equipped with a touch screen and theoretically could be adapted to recognize such characters “written” on the watch face.

Note that both the notation and the device must have the correct affordances to support heads up writing. With our wristwatch example, the watch bezel would provide a tactile reference to constrain the finger to the surface of the watch crystal, so heads up writing would work. However, with the Palm Pilot, for example, this is not the case. Even though both might use Graffiti, with the Pilot, one must be visually attentive to ensure that strokes are made in the appropriate part of the screen. Otherwise, one might try to enter text on the numerical region, or in the data display/selection region.

Finally, it is also important to note that the concept of heads up writing predates any of the alphanumeric single stroke alphabets. In the 1980's, Casio made a watch, the AT-550, illustrated in Figure 32, and a pocket organizer, the PF-8000, illustrated in Chapter 2, that supported this, despite the fact that they did not utilize single stroke character entry. Single stroke alphabets make it easier to implement, but if one is clever about how character segmentation is done, they are not a prerequisite.





Casio DBC-150-1 Databank Casio AT-550 Finger Trace Calculator Watch

**Figure 32:** Two Wristwatch-Based Electronic Organizers

*Illustrated are two wristwatches with built-in calculators from the same company, Casio. With the one on the left, data is entered using the integrated mini keyboard. The AT-550, shown on the right, has a touch screen mounted over the its face. With this design, users were able to enter numbers by tracing them on the face of the watch, using their fingers. The bezel of the watch provided tactile boundaries on the drawing space. This approach would lend itself quite well to single-stroke shorthand techniques, like Graffiti. What is most impressive is that this watch was released in January of 1980. It was way ahead of its time, and is a cousin to the Casio PF-8000 Databank, illustrated in Chapter 2. (Left photo courtesy of Casio Inc.)*

We have gone into a fair bit of detail in our discussion of single stroke text entry systems. Part of the reason is simply that they are interesting and illustrate a lot of ingenuity. Another is that with the increasing adoption of small PDA-type appliances, this type of technology is of growing practical relevance. Third, these systems also provide a rich set of examples that can serve as case studies and objects of comparison for the student. *Quickwriting* claims to be "several times faster than *Graffiti,"* for example, yet there are no studies to support this. Should you believe the anecdotal claims? How can we apply what we learned in Chapter 7 about time-motion studies, for example, to do some "back of the envelope" or even more formal comparative estimates of performance? For a typical English sentence, is there, in fact, any saved motion in writing cursively with *Quickwriting*? Or, how would you determine where the balancing point is between ease of learning *vs* the speed that one can attain once learned? As we get further into this book, we can start to apply what we have learned earlier to the problems that we now encounter.

Finally, in the "there's nothing new under the sun" department, it is interesting to conclude our discussion of single stroke shorthand with a brief historical notes. It turns out that this class of shorthand is a rater old idea, invented around 63-58 BC by a freed slave of Cicero, *Marcus Tullius Tiro*. His shorthand, *notae Tironianae*, or *Tironian notae,* illustrated in Figure 34 was used for about 1,000 years, among other things, to record the minutes of the Roman Senate (Gaur, 1992).

> But during the Middle Ages – perhaps because scholarship in general fell into disrepute – shorthand became associated with evil spirits and witchcraft, and a person employing it was believed to be possessed by the devil. (Panati, 1984, p. 81)

Given how devilishly difficult today's computers are to use, one could argue that little has changed since the 12th Century! Just look at the comparison between Natew Tironianae and Graphitti in Figure 33.

The real lesson in this example, however, is to recognize the value of scholarship in design and innovation. Invention has as much to do with understanding and drawing on the past as it does creating the new, something that seems too often forgotten.

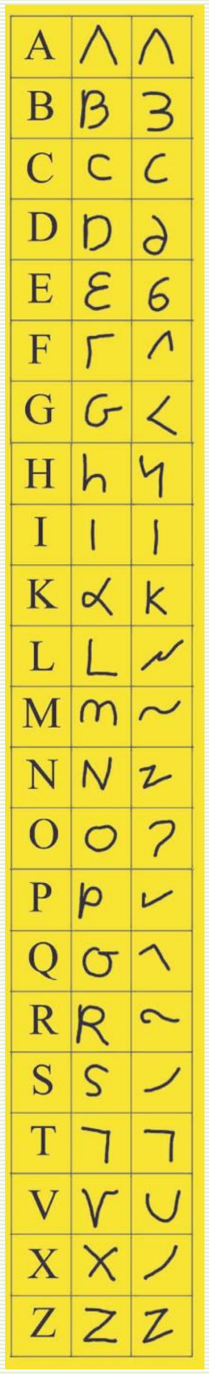

A
B
C
D
E
F
G
H
I
K
L
M
N
O
P
Q
R
S
T
V
X
Z

**Figure 33:** Comparing Roman, Graphitti and Notae Tironianae Scripts

*Beside each modern alphabetic character appear the Graffiti and Notae Tironianae, symbols that represent it (middle and right columns, respectively). Notae Tironianae, likely the first singlestroke short-hand, was developedin 63 BC by a freed slave of Cicero.* (From Buxton, 2005).

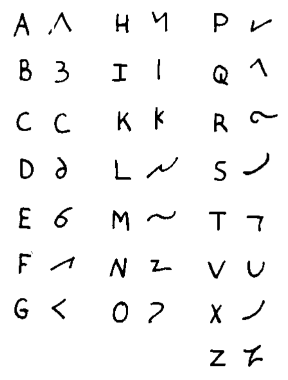


Figure 34: *Notae Tironianae,*

*Likely the first single-stroke short-hand, developed in 63 B.C. by a freed slave of Cicero. (Illustration from Panati, 1984, p.81)*

The User: Learning and Training
• tie in to speed (previous section)
• not self-revealing (in contrast to menus)
• size of symbol set
• exploit generic operations?
• design of symbol set
• like icons
• representative or not
• drawing speed vs mnemonic
• transfer: what do people do
• studies eg., IBM Wolf (1986), Rhyne & Wolf (1986), Wolf & Morrel-Samuels (1987) Wolf (1988), Welbourne & Witrow (1988).
• include figures (such as p366 in HCI '88)

## Why the Competition? Typing on a Graphical Keyboard & Stroke Recognition

All of the examples in the previous section pitted tapping on a graphical keyboard against approaches where strokes, which frequently involved leaving a trail of digital ink, were recognized (often in combination with some kind of menu system). Both approaches have a legitimate place and are worth studying. But ….

Speak about gesture keyboards:

- Buxton, W. & Kurtenbach, G. (2000). Graphical keyboard. *US Patent 6,094,197,* July 25, 2000.
- SHARK, ShapeWriter, etc. including precedent of: Nantais, T., Shein, F. & Treviaranus, J. (1994). A Predictive Selection Technique for Single-Digit Typing With a Visual Keyboard. *IEEE Transactions on Rehabilitation Engineering,* 2(3), 130-136.
- Isokoski, Poika (2004). Performance of Menu-Augmented Soft Keyboards, *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'04)*, 423-430.

### Form Filling

• boring, but common application
• well suited, since based on spatially distinct cells, or fields to be filled in

**Document Creation, Editing, Annotation**

• Nugent, W.R. & Buckland, L.F. (1967), Coleman (1969), Doster & Oed (1984), Welbourne & Witrow (1988).
• figure/ground relationship
• deferring action, eg., mail
• GEdit

Selection Techniques
- text/symbolic
- all DM
- point
- drag through
- circle/lasso
- free motion (state 1) means noncontiguous regions easier

White Space
- graphics & text are different
  - fill in
  - feedback, eg., dragging

**Digital Ink and Document Morphology**

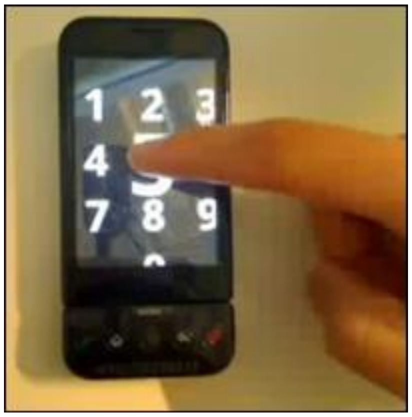
Introduce AhHa! And PARC stuff on document recognition.

**Radial Menus**

Introduce here

Discuss Pixie and Don Hopkin's work.

A recent example of using radial menus can be found in Google's mobile phone. It has far more resemblance to a phone keypad than a radial menu, but a radial menu it is. I



The radial menu is centred where you first touch, that is, the "5" always appears below where you first touch the screen. If you then release your finger without moving, a "5" is entered. The way that you enter a different digit is to move your finger in the direction of that digit, and then release it. For example, if you slide up to the right and release, you enter a "3", while sliding to the left enters a "4". The gesture recognizer, for that is what it is, can also distinguish long strokes from short ones, so a short downward stroke followed by a release enters an "8", while a downward stroke followed by a release would result in a "0" being entered.

What is strange is that they do not support the full key-pad. For example, if they included long SW and SE strokes, they could have also supported the "*" and "#" respectively.

**Marking Menus**

Introduce here

**Brown University Sketch System**

Introduce for 3D graphics

**Shared Drawing**

- Minneman & Bly (1991).
- Tang & Minneman (1991)
  - also relates to next chanpter: gestures
- Wolf, Rhyne, et. al (1991)

The System: Trainability and Extensibility
• Trainability concerns individual differences
• Extensibility concerns tailorability
• Different but related issues
• user dependent/independent
• one does not imply the other: repertoire may be fixed.
• 1st trainable char rec: Teitelman (1964)
• besides training, there can be profile setting. e.g., Apple Newton: slect patterns that most match your way of writing, but you must select from existing repertoire. Recognizer does not learn from your script over time or training.

**Transducer Technology**

• generally stylus, but depends on bandwidth
• technology really affect performance
• issues of hardware discussed in: Leedham, Downton, Brooks & Newell, 1984 / Ward and Philips (1987) / Kim & Tappert (1984)
• see Francik & Akagi (1989) for discussion of human-factors of designing the tablet and stylus of the Wang *Freestyle* system.
• State 0 transitions key for delimiting
• delimit strokes

direct/indirect?
• write directly on display, on paper, on tablet?
• hybrid with keyboard?
• could interfere with touch screen on surface (eg., Dillon)

stylus:
• tip switch
• cordless?
• states
• tip switch

Tablet / writing surface
• Ward et al
• linearity, etc.

**Recognition Technology**

Three types:
- template matching
- feature extraction
- connectionist
    - Martin, G.L. & Pitman, J.A. (1989)
    - Le Cun et al (1989). neural net on chip for numeral recognition. Trainable.
  - Pitman (1991)

- Defer: not an AI or signal processing book

- use of temporal cues?
- can help
- context of when a sign was specified can give important clues to interpretation of intent
- but if prerequisite, mitigates against deferred recognition

**Implications on Underlying Software**

- parallel scanning & semantic processing
- GEdit example

**The Future?**

- Pens that remember what you write, even when writing on paper. See Nabeshima et al (1995).

**Summary and Conclusions**

... the technique that is possibly the most powerful and general of all, and that certainly presents the most interesting issues to the programmer.

Newman & Sproull 1973, p. 227

Could be finally reaching time: combination of MIPS, small displays and transducers. Currently 4 portables on market. Becoming more real (Thorell, 1987). But while never(?) use 2 pencils at a time, we seldom write or draw without using our other hand. w/o that, not paper-like, and misses full potential.

Line Driven Input Video Examples

1. Software Control at the Stroke of a Pen, Pencept Inc.
SIGGRAPH Video Review 18
Pencept is one of the better (and better known) manufacturers of character recognition technology. This tape does not show them to best advantage. Their recognizer does not require training, yet can recognize a wide range of printed characters. (On the other hand, it does not permit training, so adding new characters cannot be done by the end user). The system consists of software bundled with a tablet. On the one hand, this means that the display and input surfaces are different. On the other hand, it supports dialogues that involve paper forms. Characters can be printed anywhere on the tablet. They are not restricted to the confines of a boxed grid, for example (although, you may do this if for some reason you want to).

Besides recognizing the character printed, the *size* and *position* of the characters. This is extremely valuable. For example, in a CAD program, it permits labels, their size and position to all be specified in one chunk in a way natural to the task and user. Strangely, this feature has not been exploited by any application that I am familiar with. Very strange. Seems like an opportunity waiting to happen. What I'd like is to go one step further: to enable pressure, for example, to determine wether the characters are to be bold, and slope to determine if italics are intended.

2. Paper-Like Interface, IBM.
SIGGRAPH Video Review ?
This is perhaps the best demo of character and (limited) gesture recognition being shown today. Note the use of position and size information in specifying the mathematical formula. Unlike the Pencept system, the system uses *direct input.* That is, one writes directly on the display surface.

Also unlike the Pencept system, the system makes use of gesture, as well as character, recognition. This is seen in the (seemingly compulsory) Lotus 123 example. This is a good tape, and I believe that it defines new standards.

**3. Freestyle.** Wang Labs.
SIGGRAPH Video Review 45 (1989)
Despite the obnoxious nature of this tape, the system itself is an outstanding example of economical but effective design. It is a good example of how much one can achieve with very little. It works only on "dumb" bit-maps (actually, tiff files). One can marking, send and retrieve documents with hand written/drawn annotation (and with voice). However, there is no recognition capability. All actions associated with symbols (such as "delete" or "move" must be manually executed. Furthermore, they can not be executed (even manually) on the document in which they appear (which is, as we have stated, just a bit-mapped snap-shot of the actual document.

The power of Freestyle comes largely from two sources: (1) integration with the mail system, (2) the strength of the figure-ground contrast of hand annotations on top of "typeset" material. It is an example of careful design giving rise to a lot of benefit.

The problem with the Wang system is that it is so successful (in one way) that you think that it has more smarts than it actually does. Consequently, this leads to additional expectations which cannot be met. You get let down when you find that it can't do something that, by extension, you would expect that it could. Further complicating things for Wang is that, by taking the approach that they have, they have boxed themselves into a corner: there is no smooth evolutionary path for their product to grow to a "smart" system.

Notes
Note: this chapter is incomplete. Outline of contents is given.

integrate:
Hornbuckle, G.D. (1967). The computer graphics user/machine interface, *IEEE Transactions on Human Factors in Electronics*, 8(1), 17 - 20.

Meyer, A. (1995). Pen computing: A technology overview and a vision. *SIGCHI Bulletin*, 27(3), 46-90.
Bob’s stuff from brown

Brown, C.M., 1988. Comparison of typing and handwriting in "two finger typists". *Proceedings of the 32nd Annual Meeting of the Human Factors Society*, 381-385
- note, may also be referenced in *Text Entry* in Chapter 2.

http://www.amug.org/amug/sigs/newton/nanug/PenReport/NewPenCom.html

Include stuff like Teddy 2D sketch -> 3D geometry.
http://www-ui.is.s.u-tokyo.ac.jp/~takeo/teddy/teddy.html

Sacks 3Draw @ MIT

Baudel Ligne Claire

P. Brandl, C. Forlines, D. Wigdor, M. Haller, and C. Shen, 2008. "**Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces**," in *AVI08: Proceedings of the working conference on Advanced Visual Interfaces*, New York, NY, USA, 2008, pp. 154-161.

Wobbrock, J.O., Myers, B.A. & Kembel, J.A. (2003) EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion. *Proceedings of the 16th Annual ACM Symposium on User Interface Technology (UIST'03)*, 61-70.