SYSTEMIC IMPLICATIONS OF LEAP AND AN IMPROVED TWO-PART CURSOR: A CASE STUDY

Jef Raskin

Information Appliance, Incorporated
3530 West Bayshore Road
Palo Alto, CA 94303

ABSTRACT

The lowly text cursor is a non-issue for most interface designers. Nonetheless, current text cursor designs suffer from at least two problems: one-off errors and a lack of visibility of function. These problems are exacerbated in an editing environment which uses the extremely fast Leap cursor-moving technology.

This paper presents solutions to these cursor design problems and reveals the surprising way many other aspects of system design can be improved as a consequence of designing the cursor correctly.

KEYWORDS

Cursor, dual cursor, mouse, Leap, text editor, word processor, user interface, blind.

INTRODUCTION

Cursors on computer screens, like doors on buildings [1], are so commonplace that we tend to take them for granted. However, most present cursor designs inaccurately convey their intended function. The improved cursor design presented here makes learning to use a screen-based system easier, decreases the difficulty of using a system even for experienced users, and has led to unexpected and beneficial changes throughout the design of a number of products.

The function of a cursor is to indicate the locus of action of an event. In text, it usually is used to indicate where the next character to be typed will appear, where a block insertion or other action will take effect, and where a deletion will occur if an appropriate key is tapped.

The most elementary use of a cursor is in simple typing. Surprisingly, it is here that conventional cursor design first fails.

THE CURSOR IN TYPING

A cursor in text takes the form of a (usually blinking) underline beneath a character, a box around the character, or — in some bit-mapped graphic-based systems [2] — a vertical line (which may be decorated in various ways) between two characters.

The elementary functions in typing with a display-based system include (i) tapping [3] a key in order to insert (or overwrite) the key's associated character at the location of the cursor, and (ii) a backspace function, which deletes the most recently typed character. The frequency with which humans err makes backspace an important function, but it also means that there are two loci of action in the text: the location on the display where the next character will appear if a character key is tapped; and the character on the display which will be deleted if the backspace key is tapped. These are generally *not* at the same position on the display.

Nonetheless, conventional cursors indicate only one of these two positions. This, naturally, causes confusion in beginners and errors in more experienced users. For example, consider the string

abcde

positioned somewhere on the display. The user wishes to insert the character "x" at the position currently occupied by the letter "c" in the example. With most present systems (in insert mode), the cursor is moved to the "c" — as shown by the underline in the example, and the "x" typed, yielding

abxcde

If, however, the user wishes to *delete* the "c," then the cursor must be moved to the "d"

abcde

and the backspace key tapped. This requirement for aiming "one-off" from a given target for deletion is inherently con-

fusing. Ideally, the user should aim exactly at the target of an intended operation. To alleviate this problem, some systems provide a "forward delete" function that erases the character at the cursor, thus solving the "one-off" problem. But now there are two ways to delete characters, and the user must choose between them, introducing a new possible error: using the incorrect delete function. Observation of users of such systems reveals that this error is not uncommon. While forward delete may be valuable for other reasons, it is not a solution for the problem being discussed here.

A more recent class of systems uses a between-character cursor. This cursor is based on work done in the 70's at Xerox® PARC and since embedded in popular systems such as the Apple® Macintosh™ [4]. Part of the intent of this cursor is to indicate that a new character will be inserted conceptually [9] between two existing characters. However, the one-off problem still exists. The user must remember to place the cursor to the left of the "c" in order for, say, an x to be inserted at the location of the "c" and the "c" moved to the right.

```
ab|cde
```

type "x"

```
abx|cde
```

but to place the cursor to the right of the "c"

```
abc|de
```

if one is to use backspace to remove it, resulting in

```
ab|de
```

While I have characterized this problem of cursor design as one where the user inadvertently aims at the wrong point, it is also partly a problem of visibility [1], since there is a function of the cursor that is not visible.

SEEING THE PROBLEM

The very familiarity of cursors made it especially difficult to detect the one-off problem. I was surprised, in the early stages of testing other aspects of one of our interface designs that subjects were making the one-off errors described above. It would have been an easy matter to push the observations aside and concentrate on "bigger" issues; many other designers have accepted cursors as they are. However, we were quite determined to not only bulldoze human interface mountains if we could, but to keep our users from tripping over the pebbles as well. Thus it was that in the early design stages of a commercial information appliance [5] the staff of Information Appliance Inc.™ attempted to find solutions to the one-off problem.

Before solving the one-off problem, we had already designed a two-part pointer which solved the visibility part of the problem. This pointer consisted of a blinking *cursor* (here indicated by an underline) and a reverse-video *highlight* (here indicated by **bold** type). In this example, the cursor has been moved to the letter "c".

```
abcde
```

The user is told that the blinking cursor indicates where the next character to be typed will appear, and the highlight shows what will disappear when the backspace key is tapped. This, however, did not solve the one-off problem since to delete, say, the "b", one still had to move the cursor to the "c." If one defined pointer motion in terms of the highlight, then erasure worked correctly, but insertion had a one-off error associated with it.

The problem was brought into sharp focus by our extremely fast Leap® [6] mechanism for moving the cursor to a particular instance of a target character. The mechanism, described below, is text-based, modeless, and faster than using a mouse or other pointing device. Its very speed made any one-off errors especially noticeable and annoying.

DUAL CURSOR BEHAVIOR UNDER CURSOR MOTION

Considerable debate arose as to whether it was better to have the cursor or the highlight land on the target, and this debate stifled progress for some six months (the debate also included various ways of implementing Leap, but that is another story). Eventually three of us [7], in what seemed to be yet another futile attempt to decide which of the two ways was better, simultaneously realized the solution.

In retrospect the solution seems simple, even though it was in fact far from obvious at the time: indeed, a number of people both inside and outside the company had worked on the specific problem and failed to solve it. Emotions ran high: one employee resigned partly due to his belief that we were wasting time on a "religious" dispute that had no real solution [8]. The solution was this: when the cursor is moved, we coalesce the blinking cursor and the highlight on the same target character (indicated here by an underlined bold-face character). A Leap (or other cursor move) to the character "c" in our example yields:

```
abcde
```

Now, if a character is typed, it displaces the target; if backspace is employed, the target is erased. With this scheme, the user *always* aims for the locus of the desired action and there is no one-off problem. It is clear that a similar pair of indicators could be used on a wide variety of systems. If an "x" is typed, as before, we would have

```
abxcde
```

which indicates that the next letter to be typed will push the "c" still further over, and tapping backspace will delete the just-typed "x". However, if backspace were tapped, we would have

```
abde
```

This solution turned out to be eminently satisfactory not only for beginners but even for users experienced with conventional cursors who unconsciously adapt to the new design immediately and use it without error or comment. [10]

INTERACTION WITH LEAP

The need for a better cursor design was more forcibly brought to our attention by our extremely rapid cursor moving ability. Leap is implemented with two keys, pedals or other buttons. The keys are typically accessible to the thumbs below the space bar or to the right and left of a shortened space bar. The right Leap key is used to move the cursor forward in text; the Leap key on the left is used to move the cursor backward in text. Forward and backward are defined with respect to the order in which text is usually read.

To avoid the usual pattern entry mode, a Leap key is held down *while* the pattern is typed. The cursor is immediately placed at the first character of the first occurrence of the pattern in the indicated direction. At first blush this looks like a simple modeless find, but it has a very different *feel* than other implementations with which we are familiar: the search takes place *during* the typing of the pattern, and is so fast (maximum search time is under 300 msec) that there is no apparent delay between completing the pattern and the cursor appearing at the target. Without this speed it would be hard to call Leap a cursor moving technique. Another very important point is that no delimiter is required. When users see that the cursor has appeared at the target, they merely resume typing or go on to perform whatever operation is desired. The hidden delimiter which consists of *releasing* the Leap key is so transparent and natural that even beginners rarely realize that they perform this operation.

SIMPLIFYING THE MENTAL MODEL AND WORKING WITH THE BLIND

Another advantage of not requiring an explicit delimiter is that any character may be part of a search string. In both IBM® and Apple personal computer systems, return is used as a search string delimiter which means that one cannot, for example, search for a period at the end of a paragraph (i.e. search for Period-Return-Return). In systems we design, page and document boundaries and the like all become typable, erasable, and Leapable-to characters, making for a great deal of uniformity and eliminating many special cases required by other systems.

For example, forcing a new page is thus reduced to putting in a Page character (no page commands are used). Starting a new document consists of typing the Document character. Any of these characters can be the target of a Leap, thus to move to the next page one simply Leaps to the Page character, to search the beginnings of a number of documents one Leaps to the Document character (and then can tap a Leap Again key, which Leaps to the next instance of the same pattern.) Implicit Page characters are also inserted automatically when there are enough characters to warrant a new page, these Page characters migrate as needed as text is edited. Explicit, user-supplied Page characters stay where they are typed.

The user then has a mental model where Returns, page breaks, and document boundaries are characters, each with a characteristic graphic appearance, but which behave *exactly* as do the letters of the alphabet. This mental model is far simpler than that required by systems where pages, documents and the like are unique constructs each with their own rules.

Leap has some interesting benefits besides its modelessness and ease of use: it is faster than any other cursor moving technique with which we are familiar, with expert user times averaging under two seconds [11]. Since it is context-dependent and not position-dependent, we have found that (in conjunction with an inexpensive speech synthesizer) it can be used by blind operators.

FURTHER SYSTEMIC IMPLICATIONS

To highlight (or select) text in a Leap-based system one positions the cursor on one end of the text to be selected, Leaps to the other end of the selection in either direction and taps the Highlight key. Since, in this paradigm, there is *always* a highlight, we say that when the highlight encompasses more than one character it is "extended". Note that, unlike many popular systems where the cursor position and the current text insertion point can occur at widely separated points (even to the extent that typing on the keyboard can affect text not visible on the screen), with this system the highlight is always adjacent to the cursor. Since the cursor is always on the screen, the position of at least one end of the highlight is always visible. Systems where the selection can disappear can appear mysterious to the user.

The proximity of cursor and highlight, along with the speed of Leap and the desire to avoid other confusing aspects of more typical user interfaces inspired one employee (Dr. Scott Kim) to suggest that we abandon the usual cut-and-paste method of moving text. In his improvement, Leaping while there was an extended highlight caused the selection to be inserted at the cursor's destination. A separate Copy key is used to make a copy of a selection.

Pointers are left at the end of the move operation so that if the Highlight key is tapped the moved selection is automatically rehiglighted. [12]

MOVING A BLOCK OF TEXT: COMPARISON WITH A CONVENTIONAL MOUSE-BASED SYSTEM

Apple Macintosh	*Canon Cat*
1. Select text to be moved	1. Select text to be moved
2. Use Cut operator	2. Move cursor to destination
3. Move cursor to destination	
4. Use Paste operator	

CONCLUSION

The interface ideas described here tend to be quite different from the majority of work being done in the human interface field. This work touches some low-level areas where it is taken for granted that we have long known the "right way" to do things. In fact, some of the elementary operations and conventions common to today's "advanced" systems are less

than optimal. For example, the two-part cursor described here is easier to learn and use than traditional cursors and causes fewer pointing errors. The success of the cursor design and the related Leap cursor-moving technology led to further beneficial changes in the design of some commercial multi-purpose systems. All of these systems have exceptional usability and good feel. Research is called for to further quantify and understand these effects.

NOTES AND REFERENCES

1. Norman, Donald A. *The Psychology of Everyday Things.* Basic Books, 1988.

2. Xerox PARC's Alto systems and their derivatives, e.g. most graphic work stations and Apple's Macintosh series.

3. At Information Appliance Inc. we use the term "tap" to mean the consecutive actions of depressing and releasing a key or button with no intervening actions. The term "press," often used in this connection, connotes only pushing a key or button downward. For example, we would describe the act of typing a capital letter B on most systems as follows: "Press and hold down the Shift key and, while holding it down, tap the key marked 'B'; then release the Shift key."

4. The author, who created the Macintosh project at Apple Computer, and who had been a frequent visitor to Xerox PARC (Palo Alto Research Center) in the early '70's, introduced the Xerox paradigm to Apple. It should be noted that those elements of the work done at Xerox PARC which Apple used were not "pirated," but were used with permission as the result of negotiations between Apple and Xerox.

5. The best-known product to utilize the concepts presented here is the Canon Cat™ "work processor." Its scope of application includes word processing, data base, spreadsheet, telecommunications, programming, and calculation. Based on a Motorola 68000 CPU, the product had a list price of $795 at the time this was written.

6. Leap is a registered trade mark of Information Appliance Inc. Leap technology and the cursor design described here are covered by patents and/or patents pending. The screen appearance and the look and feel of the interface are protected by copyright. Licenses to use this work are available from Information Appliance Inc.

7. It was truly simultaneous: one second we were discussing the situation and then something in an apparently irrelevant comment one of us made must have triggered an identical thought in each of us. We all laughed and knew the problem was solved without saying a further word. The participants were Dr. James Winter, Dr. Renwick Curry, and the author.

8. It is hard to decide when to stop working on what seems like an intractable problem. The meeting alluded to above was called by the author since he had an intuition — nothing more — that a solution was still possible and would come soon even though a concerted effort for months had failed. This is, of course, the kind of unscientific thing that goes on all the time even in the most technical of environments.

9. In fact, the new character is not inserted between two characters, but when the cursor lies between two characters the new character is inserted on top of the character to the right of the cursor, and the character that was to the right of the cursor moves out of the way to its right. This form of cursor fits a mental model of text as a sequence better than it represents what actually happens on the display. On the Macintosh, if you try to put the cursor on a particular character it slips off to the right or left, depending on whether you are to the right or left of the center of the character.

10. Being a small company, we do not have the resources to to a good quantitative study on the decreased error rate due to the new cursor design. However, so much of our work has proved so effective in practice and user satisfaction that it begs for proper academic research to either confirm or deny our conclusions, to explain and possibly extend our successes, and to quantify the results. We are always happy to cooperate with researchers in this regard.

11. One subject, a professional writer with good typing skills, after over three months experience with Leap, achieved an average cursor moving time of 1.34 seconds including Leaps to targets that were not on-screen. The times were measured in the process of ordinary editing. We observe that this is considerably less than the 2.02 seconds average time for the use of a mouse by an experienced user for on-screen moves cited in Card, Moran, and Newell, *The Psychology of Human-Computer Interaction,* Lawrence Erlbaum Associates, 1983, pg. 237. Since our time measurement was from initial press of a Leap key to the resumption of typing rather than to the moment the cursor lands on target, we have added (in accordance with the techniques given in the cited work) an extra homing time to the mouse time to make the results more directly comparable. The advantage of Leap over a mouse in moving to off-screen targets (where the mouse-based system must resort to scroll-bars or other special techniques) is considerably greater.

12. In the Canon Cat, the text was left highlighted after the move, on the grounds that the user might want to move it again. In practice this design error resulted in many unwanted secondary moves.