DECOUPLING POINTER AND IMAGE FUNCTIONS OF CURSORS IN SPACE, TIME, AND AVAILABILITY

Michael J. Muller

Bellcore
444 Hoes Lane
Piscataway, NJ 08854 US
(bellcore!ctt!mjm)

ABSTRACT

A Decoupled Multifunctional Cursor (DMC) is described which separates the pointing function of a cursor from the information-carrying function of a cursor. Decoupling in time occurs through maintaining pointer visibility at all times, while providing image visibility only upon need. Decoupling in space occurs through manipulation of the relative positions of the pointer and image. Decoupling of availability and visibility occurs through the user's ability to execute functions even if the image is invisible. This display technique has a number of advantages over conventional cursor usage, and over conventional pop-up menus, including utility within adaptive user interface paradigms.

KEYWORDS: Direct manipulation, cursor, syntax, chord, mouse.

RÉSUMÉ

On décrit un Curseur Multifonctionnel Decouplé qui sépare la fonction de pointeur de la fonction porteuse d'information d'un curseur. Le découplage temporel s'effectue en conservant une visibilité constante du pointeur tout en ne maintenant la visibilité de l'image que lorsque celle-ci est desirée. Le découplage spatial s'effectue en manipulant les positions relatives du pointeur et de l'image. Le découplage de la disponibilité et la visibilité provient du fait que l'utilisateur peut exécuter des fonctions même lorsque l'image n'est pas visible. Cette technique de visualisation offre un certain nombre d'avantages par rapport à l'utilisation d'un curseur conventionnel ou par rapport aux menus instantanés conventionnels, entre autres son utilité pour des paradigmes d'interfaces-utilisateur adaptifs.

MOTS-CLÉS: Manipulation directe, curseur, syntaxe, corde, souris.

Introduction[1]

Direct manipulation user interfaces frequently use a graphical cursor (for critical reviews, see Buxton [1]; Hutchins, Hollan, and Norman [2]; Muller [4]). The cursor contains two distinct functionalities which are often confounded:

- **a pointer,** for selection of data objects
- **an image,** for representation of the current operation which will be performed if a data object is selected

Several recent efforts have considered these two functions separately (Marcus [3]; Muller [4]; Myers [5]; Myers and Buxton [6]; and Smith [8, 9]).

This paper extends the previous work, explicitly decoupling the pointer function from the image function in space, time, and availability.

Confounding the Functions

The simplest graphical cursors serve as pointers only. Examples are shown in Figures 1A-B. Each pointer has an associated hot spot (Figures 1C-D), which is the exact pixel location at which the pointer points -- i.e., a data object beneath this particular pixel receives the action of the cursor.

The shape of the cursor is often used to indicate different modes of the system. For example, a wait-state can be indicated by a hand in a "stop" gesture (palm held up toward the user), or by an image of a clock or watch.[2] In graphic arts support systems, the cursor may be represented as a drawing instrument -- e.g., a

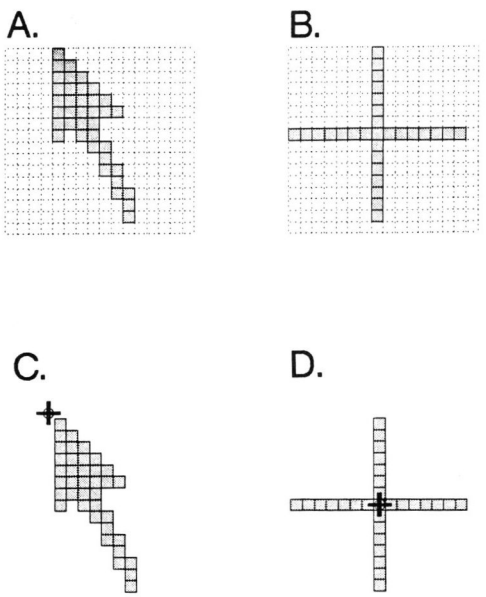

Figure 1. Two simple cursor shapes with pointing functionality only. A. An arrow pointer. B. A cross-hair pointer. C. The arrow pointer with its hot spot indicated by tiny, circled cross-hair. D. The cross-hair pointer with its hot spot indicated by tiny, circled cross-hair.

1. The opinions in this paper are those of the author, and do not necessarily reflect those of Bell Communications Research.

2. The names of products or their suppliers will be omitted for antitrust simplicity.

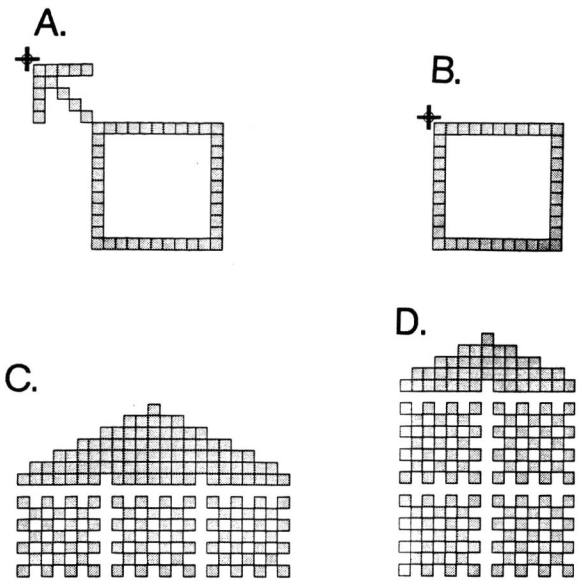

Figure 2. Unconfounding pointing functionality from image functionality. A. An unambiguous pointer frame with an enclosed image (redrawn from Marcus [3]). B. An ambiguous pointer frame with an enclosed image, in which the upper left corner serves as a pointer by convention (redrawn from Marcus [3]). C. A Multifunctional Cursor for a three-button mouse (redrawn from Muller [4]). D. A Multifunctional Cursor for a two-button mouse with single-click and double-click protocols (redrawn from Muller [4]); the top row is for single-clicks, and the bottom row is for rapid double-clicks.

crayon, which leaves a thin line image from its point, or a brush shape of specified length, breadth, and orientation.

However, the practice of representing the state of the system in the shape of the cursor can lead to problems. Unlike the images from graphic arts support systems, some cursor shapes can be very ambiguous pointers. The need to represent information in the cursor shape thus conflicts with the need to make the cursor shape a clear pointer.

Unconfounding the Functions

Marcus's approach to this problem was to enclose the informational image within a frame (Marcus [3]). Some of Marcus's frames were unambiguous pointers (e.g., Figure 2A); others used a convention of upper-left-corner-as-pointer (e.g., Figure 2B). Muller's Multifunctional Cursor (MC) [4] extended this approach to include multiple-button mice with multiple, distinct image regions for each button (Figure 2C), and for each button protocol (e.g., the single-click/double-click cursor image of Figure 2D). Myers [5], Myers and Buxton [6], and Smith [8, 9] provided a separate cursor/pointer functionality which could manipulate information icons -- an image of mouse in Myers' PERIDOT, or a variety of tool icons in Smith's Alternate Reality Kit (ARK).

Further Unconfounding

This paper extends the approach of Muller [4] and of Myers [5] and Myers and Buxton [6]. In the work reported here, the pointer function and the image function are explicitly decoupled from one another in space and time. They nonetheless function as a single cursor.

DECOUPLED MULTIFUNCTIONAL CURSOR

Figure 3 shows an example of a Decoupled Multifunctional Cursor (DMC) for use with a three-button mouse using both single-click and double-click protocols. The DMC consists of two parts: a pointer region and an image region. The pointer region is capable of continuous movement across the screen, under the control of a mouse. The image region is not capable of continuous movement: instead, it is rapidly relocatable.

The relocation algorithm works as follows:

1. When the pointer region ceases movement, a count-down timer begins to tick. When the timer reaches zero, the image region appears adjacent to the pointer region.
2. When the pointer region is in motion, the image region is made invisible.

The image region is displayed and removed through bitblt operations, which are also used to preserve the contents of the screen which are overwritten by the image region. Bitblt operations are similarly used to restore the pre-image contents of the screen.

Decoupling in Time

The pointer is always visible. The image is sometimes visible and sometimes hidden. This constitutes the decoupling of pointer and image in time.

Decoupling in Space

When the image is made visible, its default position is to the right of the pointer, with its upper margin at the same horizontal coordinate as the point of the arrow. However, this convention can lead to problems: If the arrow is moved to the extreme right or the extreme bottom of the display, then the image will not fit onto the screen -- or, in some implementations, an error condition will result as the system attempts to draw the image outside of the screen boundaries.

Therefore, the relative positions of pointer and image are modified to insure visibility of the image. At the extreme right screen margin, the image appears to the left of the pointer, and

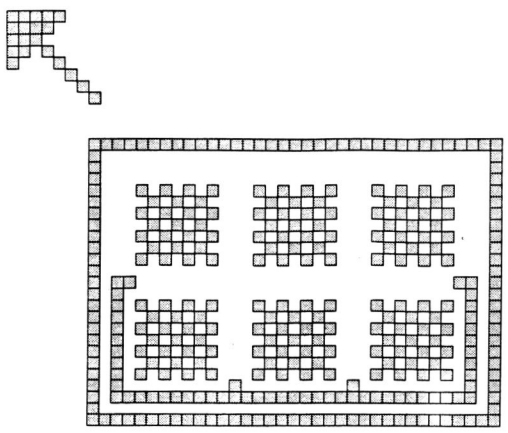

Figure 3. An example of a Decoupled Multifunctional Cursor for a three-button mouse with single-click and double-click protocols. The pointer region (upper left) and the image region (center) move separately from one another, but function as a single, integrated cursor. The checkerboard patterns indicate that no functions have as yet been loaded into the image.

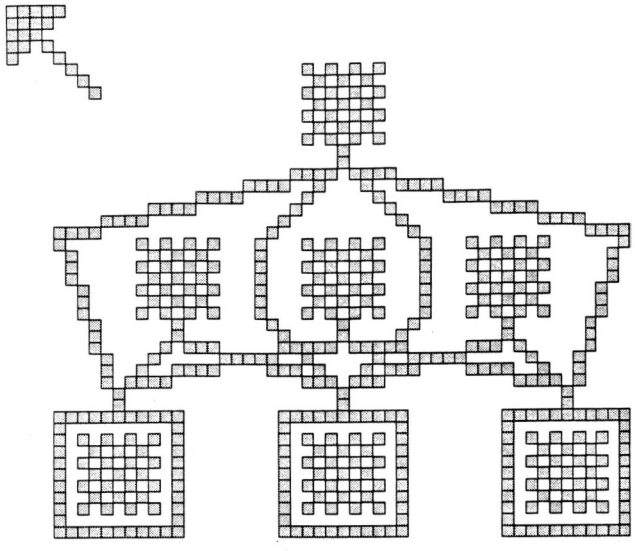

Figure 4. An example of a Decoupled Multifunctional Cursor for a three-button mouse with single-click chord protocols. The bottom row corresponds to single-button chords (L, M, and R from left to right, respectively). The middle row corresponds to two-button chords (LM, LR, and MR, respectively). The top row corresponds to the only possible three-button chord (LMR). The following functions have been loaded into the cursor image: L=edit (an eye, for "look at the file); M=rename; R=copy; LM=C compiler ("cc"); LR=linker ("lk"); MR=debugger (a magnifying glass, for close examination); LMR=erase (a trash can).

so on. This modification constitutes decoupling of pointer and image in space.

Decoupling in Availability

Depending upon whether the pointer is in motion, the image may be visible or invisible. However, the visibility state of the image has no effect upon the availability of the functions which are activated through mouse button clicks. That is, the user may execute a function on a data object through either of two sequences:

- **Full Image Visibility**
  1. Move the pointer to the data object, and then stop moving the pointer.
  2. Wait for the image to become visible.
  3. Examine the image for the icon which corresponds to desired function.
  4. Click the (visually-cued) mouse button associated with the desired function.
- **Image Invisibility**
  1. Move the pointer to the data object.
  2. Click the (remembered) mouse button associated with the desired function.

The same function is executed through either sequence. This constitutes the decoupling of image visibility from function availability.

EXAMPLE OF LOADING AND EXECUTING OPERATIONS

Figure 4 shows an example of a cursor for use with a three-button chording mouse. A total of seven chords may be entered

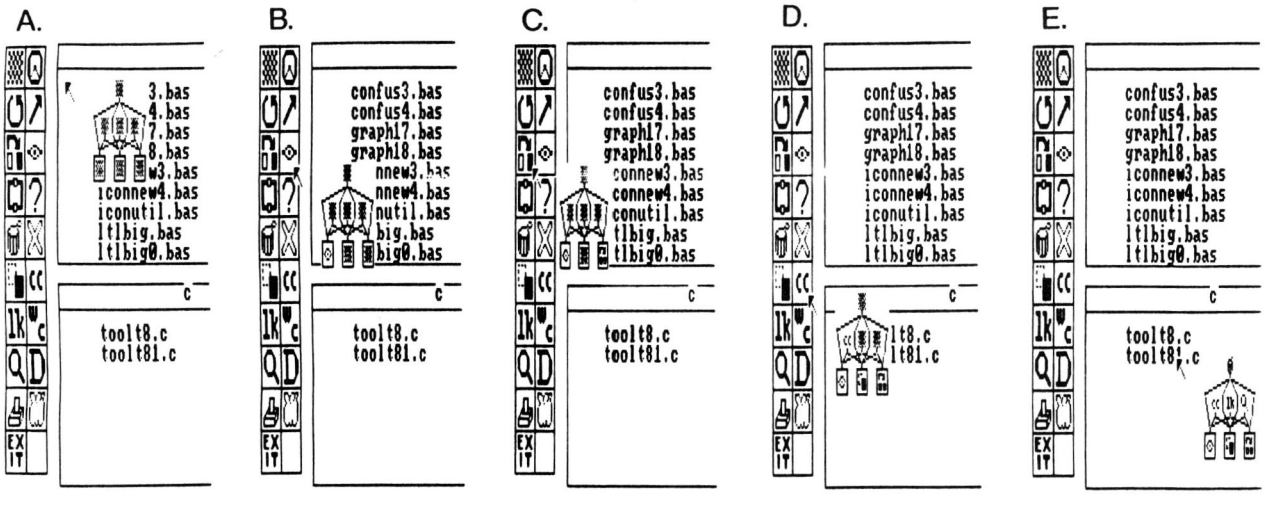

Figure 5. Example of loading functions into the Decoupled Multifunctional Cursor (DMC) of Figure 4. A. The DMC is shown before any functions have been loaded. B. Loading the edit function into the L chord (a single-button chord on the left mouse button). C. After the rename function has been loaded into the M single-button chord (not shown), the copy function is being loaded into the R single-button chord. D. The C compiler function is loaded into the LR two-button chord. E. After the linker function has been loaded into the LR two-button chord (not shown) and the debugger function has been loaded into the MR two-button chord (not shown), the erase function is loaded into the LMR three-button chord. F. The pointer region is positioned on a data object -- the file "toolt81.c" -- and the user is ready to execute one of the seven functions on that data object.

by simultaneous button-presses: three single-button degenerate-case chords; three two-button chords; and one three-button chord.

Figures 5A-E show some of the operations required to load the functions into the cursor of Figure 4. The following functions have been loaded into the cursor image:

- **L** Edit (an eye, for "look at the file)
- **M** Rename
- **R** Copy
- **LM** C compiler ("cc")
- **LR** Linker ("lk")
- **MR** Debugger (a magnifying glass, for close examination)
- **LMR** Erase (a trash can)

Figure 5F shows the cursor positioned to apply one of the operations to a data object -- in this case, a data file. The DMC has been implemented as a robust, functional prototype in an IBM-compatible personal computer environment.

DISCUSSION

Usability

Although behavioral testing is still in the future, the DMC appears to offer advantages over conventional cursor usage. The display is uncluttered by a pictorially complex cursor image, of the sort used by Muller [4] and by Smith [8, 9]. Nonetheless, the information in the more complex image is available upon need. The user makes his or her need known to the system simply by waiting. That is, a naive user who is confused into inactivity will be prompted with more information.

Expert users can bypass the additional information (and display clutter) by entering their mouse clicks or chords while the cursor is in motion; a small continuous motion is sufficient to keep the image region from being displayed.

Contrast with Pop-Up Menus

The DMC differs from standard pop-up menus (e.g., Smith, Irby, Kimball, Verplank, and Harslem [7]) in a number of respects. There is only one function available in any particular state of a pop-up menu system. This is typically the **select** function. Once a data object has been selected with a designated **select** mouse button, one of the other mouse buttons is used to pop-up a menu. The **select** button is then used to select the desired function from the pop-up menu. By contrast, the DMC provides multiple functions which are directly available by a single mouse click or chord.

Another difference is that functions are invisible in a pop-up menu system, whereas they are displayed (after a pause) in a DMC system.

Syntax of Menu Use and DMC Use

The DMC different from a pop-up menu with a temporal delay in two ways.[3] First, when the DMC appears, the user does not need to move the cursor to the image of the DMC; by contrast, when a pop-up menu appears, the user must move the cursor to the menu to select an item from it.

The second difference is more important. The DMC is based upon a simplification of direct manipulation user interface syntax called the ToolTray mental model (Muller [4]). This syntax can be summarized as **tool-object** (i.e., first select the tool, then apply it to an object), and can be contrasted to the diversity of syntaxes used in other direct manipulation user interfaces -- sometimes multiple syntaxes within a single system or product. Consider, for example, popular graphic arts support programs in which a **tool-object** syntax is used to select drawing tools, and an **object-tool** syntax is used to access menu bar operations. Many pop-up menu systems are implicitly **object-tool** syntaxes, based on their underlying object-oriented programming models. But it is arguable that many common experiences support the **tool-object** syntax, as do the previous generation of command-language user interfaces. Mixing the two syntaxes within a single environment can be confusing to non-programmer or non-expert users.

According to the ToolTray model, the user of a MC or DMC should first select the tool (by loading it into the MC or DMC image), and then apply that tool to a data object. This is the opposite of the syntax used in pop-up menus. It will be interesting to explore different temporal-delay scenarios for DMC usage and for pop-up menu usage: this will provide an opportunity to study the impact of the syntactic differences.

Potential for Adaptive User Interfaces and for User Models

A third difference is that most pop-up menu systems have relatively inflexible parameters, whereas the DMC offers delay parameters that may be easily modified as part of an adaptive user interface. Expert users may prefer relatively long delays between the cessation of cursor movement and the appearance of the image region. Naive users may prefer relatively short delays. An adaptive system might model the user's experience, and set the delay accordingly. The delay could be shortened following a user error, or if the user entered a task domain or a new computer tool with which she or he was not familiar.

Image Display Overhead

One other apparent advantage of the DMC is its compromise between information availability and display overhead (i.e., performance). Moving a large cursor and complex cursor image, (such as that used by Muller [4], by Smith [8, 9], and [in certain modes] by Myers [5] and Myers and Buxton [6]) can require substantial computing resources. The DMC reduces resource utilization by not requiring the movement of a large image. Rather, a relatively small processing penalty is paid to display a stationary form of the image region. Moreover, this stationary form is not displayed at all times -- only when the cursor has been at rest for a criterial amount of time.

CONCLUSION

The DMC appears to offer usability advantages and system performance advantages over existing cursor techniques. Behavioral testing will be begun shortly. A patent application has been filed. Research will continue in extensions of the DMC technique to a broader range of applications, and to adaptive user interface strategies.

Acknowledgements

I thank the following people for very helpful critical discussions: Mike Bianchi, Y.K. Chan, Jane Daniel, Lois Flamm, Shu-Chu Hsi, Adam Irgon, Peter Koppstein, Wei-Ching Lin, Susan Man, Bill Mansfield, John Peoples, Ruth Quigley-Lawrence, Judy Schroepfer, Mary Anne Smith, Abe Shliferstein, Danny Wildman, and Jean Zolnowski.

REFERENCES

[1] Buxton, W. (1986). There's more to interaction than meets the eye: Some issues in manual input. In D.A. Norman and S.W. Draper (Eds.), *User-centered system design*. Hillsdale, N.J.: Erlbaum.

3. I thank an anonymous reviewer for asking this question.

[2] Hutchins, E.L., Hollan, J.D., and Norman, D.A. (1986). Direct manipulation interfaces. In D.A. Norman and S.W. Draper (Eds.), *User-centered system design*. Hillsdale, N.J.: Erlbaum.

[3] Marcus, A. (1984). Corporate identity for iconic interface design: The graphic design perspective. *IEEE Computer Graphics and Applications 4* (12), 24-32 (December 1984).

[4] Muller, M.J. (1987). Multifunctional cursor for direct manipulation user interfaces. In *Human factors in computing systems*. Washington, D.C.: ACM, in press.

[5] Myers, B.A. (1987). Creating dynamic interaction techniques by demonstration. *Human factors in computing systems and graphics interface*. Toronto, ONT.: ACM SIGCHI.

[6] Myers, B.A., and Buxton, W. (1986). Creating highly-interactive and graphical user interfaces by demonstration. *ACM SIGGRAPH 20*, 249-258.

[7] Smith, D., Irby, C., Kimball, R., Verplank, W., and Harslem, E. (1982). Designing the Star user interface. *Byte 7* (4), 242-282.

[8] Smith, R.B. (1986). The alternate reality kit: An animated environment for creating interactive simulations. *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages*. Dallas, TX.: IEEE Computer Society.

[9] Smith, R.B. (1987). Experiences with the alternate reality kit: An example of the tension between literalism and magic. *Human factors in computing systems and graphics interface*. Toronto, ONT.: ACM SIGCHI.