

CrossY

A crossing based drawing application

Georg Apitz

François Guimbretière

Department of Computer Science
Human-Computer Interaction Lab
University of Maryland,
College Park, MD, 20742
{apitz, francois}@cs.umd.edu

ABSTRACT

We introduce CrossY, a simple drawing application we used as a benchmark to demonstrate the feasibility of using goal crossing as the basis for a graphical user interface. We show that crossing is not only as expressive as the current point-and-click interface, but also offers more flexibility in interaction design. In particular, crossing encourages the fluid composition of commands making it easier to create more fluid interfaces.

While crossing was previously identified as a potential substitute for the classic point-and-click interaction, this work is the first to report on the practical aspects of implementing an interface based on goal crossing as the fundamental building block.

INTRODUCTION

The recent introduction of portable, pen-based computers has demonstrated that, while very powerful, the standard WIMP-interface (Windows, Icons, Menus and Pointing device) is not very well adapted to direct pen interaction. Many WIMP interactions that were originally developed for the mouse are difficult to perform with a pen on a tablet computer. A prime example is the double click: while easy to perform in a mouse environment (since the pointer is stable), it proves to be quite difficult in pen-based interfaces. Other difficulties that arise in pen-based interfaces include occlusions created by the user's hand due to the direct setting, difficulties in using modifier keys (such as pressing shift to extend the current selection), and reduced access to keyboard shortcuts which are crucial for expert performance.

Several solutions have been proposed to address these problems. However, by its very nature, the design paradigm

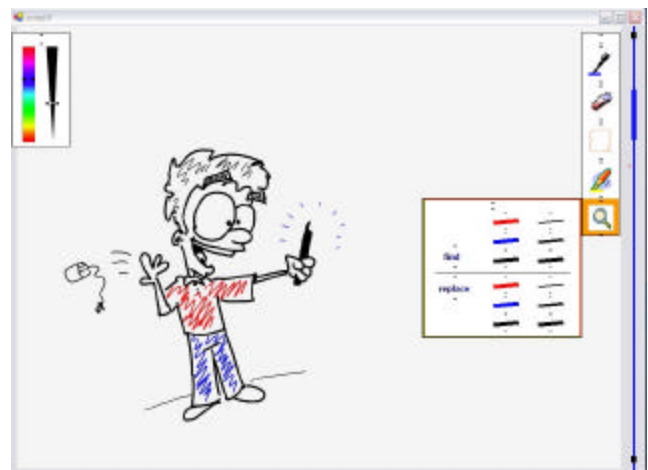


Figure 1 The CrossY interface showing the brush-palette and the palette with the find/replace dialogue open.

of current Graphical User Interfaces (GUI) is not well adapted to the pen's natural affordance: stroking. Traditional point-and-click interfaces insist on segmenting user interactions in a sequence of point-and-click interactions. This makes using the current interface with a pen a frustrating experience as users alternate between a very natural and fluid input mode for sketching or taking notes and a very rigid and segmented interaction while using the GUI elements.

At the same time, recent experimental results by Accot et al. [3] have suggested that steering through goals is at least as efficient as pointing and clicking and could be a viable substitute to pointing and clicking. Yet, with a few exceptions limited in scope (e.g. Lotus Notes and Baudish's toggle maps [4]), designers have not explored the potential of crossing as building block for GUIs.

In this paper we present the first attempt of a systematic exploration of crossing as a fundamental building block of graphical interface interactions. We developed CrossY (Figure 1) a simple sketching application for which all interface elements, (including menus, buttons, scrollbars,

and dialog boxes) rely solely on crossing. Our work not only demonstrates the feasibility of crossing as an interaction paradigm in a real life application, it also provides initial feedback on the challenges one faces when developing such a crossing interface. We found that crossing is well adapted to both pen-based and mouse-based interactions, it is more expressive than the equivalent point-and-click interfaces, and it encourages a fluid composition of commands. We also found that, to leverage this latter advantage, special consideration of the interface's layout is required that is not as important in traditional interfaces; designers must acknowledge that the arrangement of the widgets decides which commands can be combined.

MOTIVATION AND DESIGN GOALS

While the point-and-click interface has been very successful for desktop computers, many Tablet-PC users find that it is not well adapted to pen-based interactions. In part the problem rises from the mismatch between interface and the interaction device: while the current interfaces were designed in an indirect pointing configuration with a stable pointer controller, the tablet computing focuses on a direct setting, with a pen, a noisy input device. We believe that these complaints have a deeper root: pen use encourages a fluid, continuous style of interactions based on strokes, whereas point-and-click interfaces insist on segmenting the users interaction in a series of pointing steps.

To address this fundamental issue, we decided to explore using crossing instead of pointing as suggested by Accot [3]. We developed CrossY, a simple drawing application to examine the strengths and weaknesses of crossing as the building block of interaction design. We decided to use a drawing application since it supports the fact that a pen is rather used to draw strokes than to type.

In designing a new interaction language, one is presented with many choices. Therefore, we decided to limit the scope of this early exploration by focusing on the following key aspects:

- **Expressiveness.** One of the most important question to be addressed is: *If the new language can express as rich a set of features as the language it means to replace?* To answer this question, we decided to examine how the key elements of a basic WIMP interaction could be implemented using crossing. As a starting point we decided to implement standard buttons, scrollbar, menu systems, dialog boxes (including selecting an item in a list) and a simple set of window management tools (Figure 2). In each case, our goal was first to mimic existing capabilities before moving to new features.
- **Fluid composition of commands.** As illustrated by Lotus Notes and the toggle maps system [4], goal crossing-based interfaces have great potential to

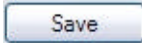

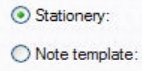

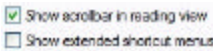

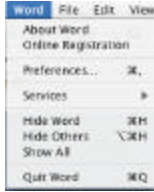

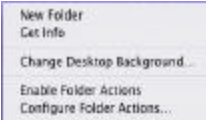
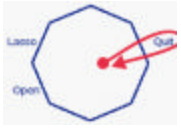



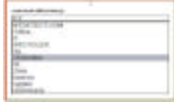
Functionality	Standard	Crossing
Buttons	Normal	 
	Radio	 
	Checkbox	 
Menu	Pull-down	 
	Pop-up	 
Scrollbar	 	
Dialog	 	

Figure 2 The correspondence table showing traditional elements of GUIs and their CrossY counterparts.

promote a fluid composition of commands. This composition allows users to issue several actions (such as selecting among a group of toggle switches) in one single stroke. Our goal was to determine if this feature could be extended to a wider set of interactions such as a search and replace task. We also examined if the advantage of transitioning from a visual interface to a gesture based interface, as developed in the Marking Menu [11], could be extended to the selection of several commands inside a dialog box

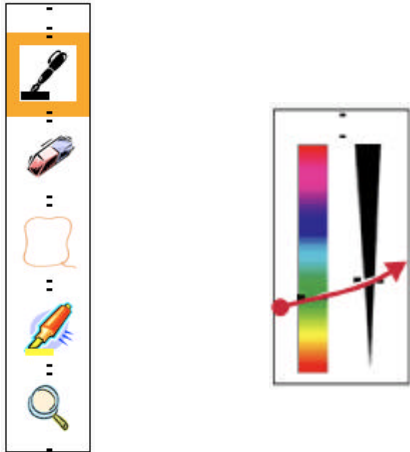


Figure 3 (left) The CrossY-palette with the feedback that CrossY is in pen-mode (highlighted pen). (right) The brush-palette and one stroke which selects color and width. Two small lines in the title bars show the crossing position to move the widgets.

- **Efficiency.** Reaching the goals above will have little impact if the price to pay is an inefficient interface. Therefore, efficiency was an important consideration during the design process.
- **Visual footprint.** Screen real estate is a valuable resource and the new interaction language should be efficient in that perspective. The case of the crossing interface is unique, since this aspect is interacting with both the composition of commands and the efficiency goals, as we discovered early during the implementation.

It is important to note that in contrast to previous conceptual explorations (such as Winograd and Guimbretière [23]), we did not focus on the creation of new, application-specific interactions. This is a deliberate choice, as focusing on standard widgets gives us a reference point against which our design might be evaluated.

PREVIOUS WORK

Several systems have departed from the strict point-and-click interface in the past. One example is Lotus Notes which lets users select several emails by pointing and clicking on the first one and then crossing through adjacent emails to select them in the same stroke. Another example is the toggle map system [4] in which users can draw on top of a set of toggle buttons to trigger them instead of being forced to click on each of them individually. Yet few have conducted a systematic exploration of crossing as a general interface design tool. A notable exception is the conceptual prototype described by Winograd and Guimbretière [23]. While Winograd presents a conceptual prototype of *visual instruments*, a full implementation of the system was never reported.

The theoretical foundation of crossing as a fundamental aspect of interface design was laid by Accot who first developed the steering law [1, 2], and then presented a more detailed analysis on how it might lead to a new interaction paradigm [3]. The work presented here leverages this theoretical basis and shows the practical aspects of developing such an interface.

Several menu systems such as Control Menu [18] and FlowMenu [7] use crossing as a way to select commands. Similar systems such as Pie Menu [10] and Marking Menu [11] use direction and pen-up transition to select commands. Our system leverages this contribution and we are using FlowMenu as our primary pop-up menu system.

In the recent years, many systems also challenged the use of a point-and-click interface for pen computing, either in whiteboard environments such as Tivoli [17], FlatLand [15] and PostBrainstorm [8], on the desktop [20], or for pen computing [21]. These systems are in general tuned to a certain class of applications (such as brainstorming for example) and did not focus on crossing as the sole interaction paradigm. They were nevertheless influential to us.

Finally several authors such as [9] have explored gesture-based interactions. While gestures are important to crossing interfaces, in general the gestures are very simple and the crossing requirement simply ambiguity detection.

CROSSY

CrossY is a simple sketching program offering several tools (such as a pen, a highlighter, and an eraser). The parameters for both the pen and the highlighter can be modified by the user. CrossY also offers a simple search-and-replace feature which lets users find strokes based on their attributes (color and thickness) and replace them. Although this drawing system is primitive, this application demonstrates how most of the standard widgets of point-and-click interfaces (Figure 2) can be implemented in a goal crossing framework. CrossY was designed to run on the Tablet-PC platform and does not require a keyboard.

Command selection

Like many applications today, the CrossY interface implements two kinds of menu systems. Common tools are accessed through the use of a tool palette (Figure 3 left) placed on the right of the display (Figure 1). CrossY offers five basic tool choices: a pen, an eraser, a lasso, a highlighter and a search tool. Each of these tools can be selected by simply crossing the icon from right to left. Since our design places the first item to cross on the farmost right side of the display, hand occlusion can be avoided (see Figure 4). The palette can also be moved around to a more convenient place for the user. To move the palette, users cross the center of the title bar between the two black marks from left to right and then the palette is attached to the pen and can be dragged around. Crossing the same area from right to left brings the palette back in its

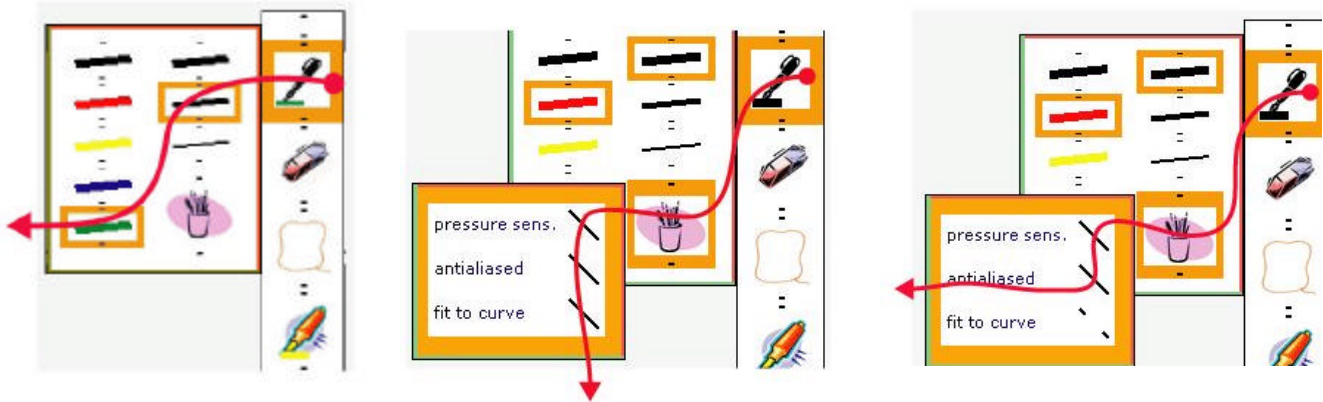


Figure 4 (left) The CrossY palette with the pen-panel opened and a single stroke which opens the pen-palette, selects width and color of the strokes and validates the selection. By convention, the left and bottom edges of each dialog box are validation edges [shown in green] and the top and right edges are cancelation edges [shown in red]. (middle) The checkbox to set the stroke-rendering attributes and one stroke selecting all items. (right). The checkbox to set the stroke-rendering attributes and one stroke selecting only two items

original position. This behavior is present for all palettes. In addition, CrossY uses FlowMenu as the primary command selection mechanism to control the application. This includes commands for lasso, open a file, save the current file and quit the application

Navigating within the document

Users navigate the document with a crossbar, the equivalent of the standard scrollbar shown in Figure 5. The crossbar looks like a simple bar spanning the length of the document viewport. It shows the beginning, the end, and the current location inside the document. To interact with it, users perform gestures crossing the bar. We provide several standard features such as page up and page down. These commands are triggered by open triangles drawn on top of the crossbar in the direction of the desired movement (see Figure 5). To issue a repeat of a previous command the user simply crosses the bar again and is now in continuous scroll mode which ends when the pen is lifted. It is important to note that, because gestures can be issued anywhere on the scrolling area, the system reduces the distance users need to travel during the preparation phase of the scrolling operation. This makes the scrolling process faster and reduces the reliance on visual feedback. To jump to a specific position inside the document, the user crosses the bar in the vicinity of the target location and then finely adjusts the position by simple dragging motions on the right side of the bar. Note that because absolute access and adjustment are now two different parts of the same interaction it is possible to provide a different gain for both phases. While the gain in the first phase is imposed by the ratio of the document length to the scrollbar length, during the adjustment phase, the gain can be reduced so that finer adjustments are possible. While some experimental scrollbars such as the FineSlider [13] provide similar aspects, this fluid integration is in general difficult to achieve in a click only interface. Another advantage of the

crossbar is that users do not need to acquire the cursor before moving to a given position in the document. Again this simplifies the overall interaction making it easier and faster to use the system.

Selecting pen attributes

In CrossY, users can select the pen attribute either by using the pen attribute dialog box or the brush palette.

Pen attribute dialog box

The Pen attribute dialog box is opened by crossing the pen tool button and extending the stroke towards the left.

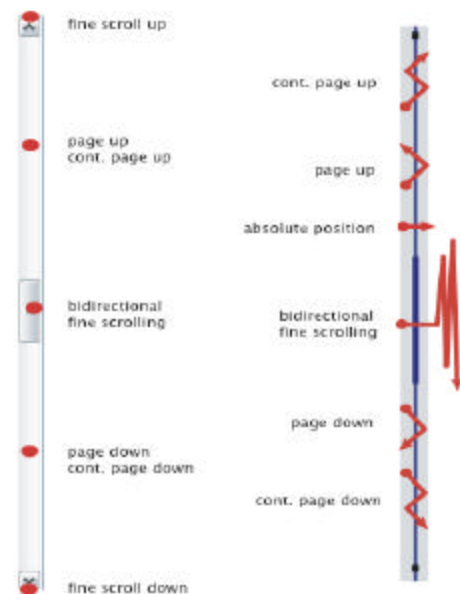


Figure 5 Comparison of the traditional scrollbar to our scrollbar. The dots indicate a click or the touching of the screen with the pen, the strokes show the gesture which triggers the action.



Figure 7 (left) Using single strokes on the find and replace panel. (middle) Combining the single strokes into one stroke. (right) Repeated find-and-replace operations using a continuous stroke.

Unlike current implementations which present “dual-use” in a tool palette (such as in Adobe Illustrator), our implementation does not force the user to dwell over the button to access the extended features. This increases the fluidity of the interaction and promotes chunking.

The pen attribute dialog box is presented in Figure 4 left. It contains a set of radio buttons used to select the size and color of the stroke. Radio buttons are designed so that crossing along the horizontal axis of the label (in either direction) will toggle the button. This feature reinforces the notion that radio buttons represent exclusive choices (Figure 4 left). By contrast, check boxes can be crossed either along their main axis or perpendicular to it. While only the perpendicular direction is needed (and reinforces the fact that the buttons are not mutually exclusive, Figure 4 middle), we noticed that it was sometimes difficult to cross only one item in that direction. Therefore we provided tilted lines as a convenience to select several items in a vertical stroke and one item in a horizontal stroke (Figure 4 right).

An unusual aspect of the dialog boxes presented in Figure 4 is that they do not seem to include an OK/Cancel mechanism. This is because the buttons are in fact very close to the edge of the window. Both the bottom and left border are validating borders (shown in green in our implementation), while the top and right border are cancellation borders (shown in red in our implementation). This layout lets users select all the options and validate the selection in one stroke.

Brush palette

The brush palette is used to set the pen attributes when a wider range of selections is desired or the exact result is not as important. The brush palette is built by setting two sliders side by side. To select a new attribute, users simply cross one of the sliders at the desired position. Again, note that the user can select different attributes in one stroke, and can memorize combinations as a certain stroke (see Figure 3).

Finding and replacing stroke attributes

Our application also provides a simple “find-and-replace” function which lets users change the attributes of

some strokes on the screen. The function is accessible through the dialog box seen in Figure 7. This form is structured around two panels. On the top panel, the user can select the thickness and color of the target strokes using a set of radio buttons. On the bottom panel, the user can select the new color and width for the selected strokes. After setting the target attributes, the user can find the next stroke forward by crossing the “find” button from right to left. In a similar way, replacement is triggered by crossing the “replace” button from left to right. While this layout seems somewhat unusual, it has been selected to encourage command composition. For example a user can in one gesture: select “medium” and “red”, cross the “find” button to find the first occurrence of this type of line, cross the “replace” button to indicate the need for replacement, and select “blue” and “thin” as the replacement values (Figure 7 middle). The command is executed as the pen is lifted from the panel. Once the parameters have been correctly selected, there is no need to reselect them, and a simple circular motion between the “find” and “replace” button will trigger the replacement (Figure 7 right). It is also easy to skip some replacements by only circling around the “find” button without crossing the “replace” button. Backwards search is provided by crossing the “find” button from left to right and an undo for the replace is achieved by crossing the “replace” button from right to left.

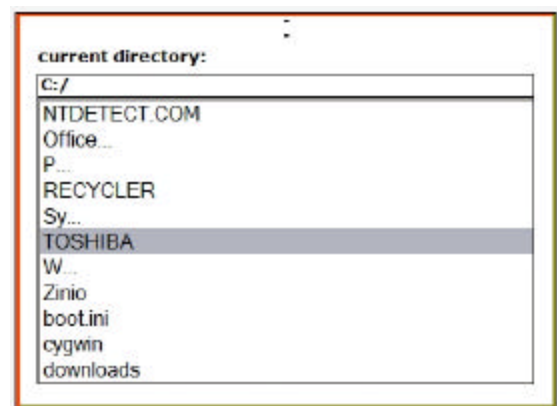


Figure 6 The file dialog box.

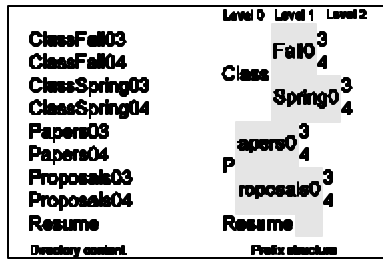


Figure 8 The creation of the file hierarchy.

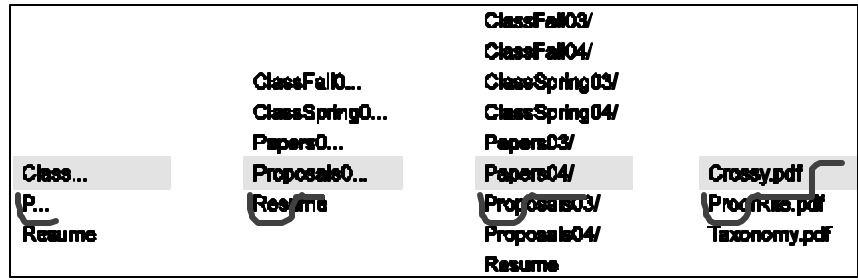


Figure 9 Navigating through a directory name presented in Figure 6 to open the file Paper04/Crossy.pdf.

Loading an existing drawing

The file dialog box (Figure 6) can be called using the FlowMenu, it allows users to navigate the file system and to load an existing drawing. At first, using crossing to navigate the file system hierarchy seems like a challenge as since current interfaces rely heavily on the use of sequential point-and-click operations for this function. Using the current standard navigation system, users first have to search through the list of files present at the current level, probably by first using the scrollbar tab for coarse adjustment, and then by moving line by line using the arrow at the end of the scrollbar. Then, they have to select the next directory to go to, (or the file to open) by double clicking on its name. For directories containing a large number of items, this method can be quite cumbersome and is far less efficient than a similar text based interface for which auto-completion makes it very easy for users to navigate even through large trees. We believe that the crossing paradigm provides ways to combine the convenience of the graphical interface with the speed of the auto-completion idea.

In our directory navigation tool, the local directory is scanned and its contents are parsed into a hierarchy of display levels. At the first level, we include all the names which are unambiguous (i.e. which do not share a common prefix with any other name) as well as the maximum common prefixes for all other names in the directory. As we go from one level to the other, for each prefix, we add the list of unambiguous names and maximum common prefixes derived from that prefix by adding in turn all possible letters following this prefix (see Figure 8). It is important to note that there are only a limited set of possible characters (256 in theory but far less in practice) that might follow a given prefix. As a consequence, moving from one level to the next only adds a small number of new options for each prefix (often less than ten). Yet, assuming an average of 10 new words per prefix, after crossing only 3 levels 1000 elements can be accessed.

Once created, this hierarchy can be navigated as follows (Figure 9): At all times the currently selected item is presented highlighted at the center of the widget. Users can change the currently selected item by dragging up and down from anywhere on the widget. To move one level

down, users simply need to make a left-to-right horizontal movement in the current gesture. This causes the current highlighted prefix (represented with ellipsis) to extend one level. To move one level up the users need to make a small right-to-left horizontal movement in their gesture. Going downward, while an unambiguous name is selected, loads the corresponding directory or file. Going upward at the root display level loads the parent directory. During navigation, feedback is provided in several ways: when the user starts a horizontal segment, a crossing goal is displayed in form of a little bar indicating the point at which the transition to the next level will be triggered. This feedback is mostly useful for the novice. For more expert users, we also generate a “click” each time a transition between levels occur and a “select” sound each time a directory (or a file is selected). To distinguish between files and directories we display a slash at the end of directory names.

This system proves to be very efficient to move through large hierarchies given that the number of levels inside a given directory is very small. This allows the user to navigate through several directory levels in the space of a small window.

Implementation

Our system was implemented on a Tablet-PC using the Windows XP Ink API and the .NET framework as the basis for our design, but could easily be ported to any other language or operating system as it relies only on basic windowing constructs (with the exception of ink management).

DISCUSSION

CrossY was implemented as a platform to investigate how crossing might improve the overall fluidity of pen-based interaction on tablet computers. While it is missing many advanced features of today’s graphical applications, it clearly shows the potential of crossing as a design paradigm. In this section we are reporting the insights we gathered while designing CrossY.

Expressiveness

From our experience implementing CrossY it is clear that the crossing paradigm is at least as expressive as the standard interface, providing the same level of functionality

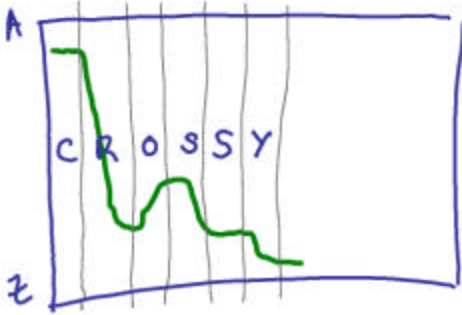


Figure 10 *Original design for the list manipulation. The system was based on an absolute mapping scheme (A on top and Z at the bottom) with a transition from one level to the next on strict boundaries shown as light lines.*

as the latter. By letting the designer take into account the position of the crossing (as in the crossbar or the pen attribute selection palette), the gesture performed (as in the crossbar), and/or the direction of the stroke, it becomes easy to provide many features with a minimal visual footprint on the screen. Our experience also provided us with some insights about the trade-offs involved while designing an interface using crossing.

Overloading versus easy discovery

The ability to overload different functions on top of the same visual artifact is certainly attractive from the designer's point of view, but it raises the problem of discovery. This is not a new problem in interface design and was identified in many systems such as the Marking Menu [12]. There are several dimensions to this problem. First, compared to pure gesture based systems (such as [9]), the crossing system provides a visual cue that some actions might be available at a specific location. If we assume the use of consistent design guidelines (such as the color-coded borders for dialog boxes), the users will acquire the basic set of overloading as they become more and more familiar with the system. This set includes aspects such as the direction used to perform an action and using the reverse direction as natural undo for the action. Furthermore, the most common functions used to manipulate the crossbar are likely to be memorized rapidly. It is also important to remember that while the WIMP interfaces provide a lot of visual feedback, the semantics of this feedback is not always clear for users. This prompted the introduction of ToolTips. The same technique might be applied here.

Fluid composition of commands

Another interesting aspect of the crossing paradigm is the possible composition of commands in one single stroke. We see this feature as a fundamental aspect of this approach since it allows users to smoothly move from novice to expert. Novice users will perform one command at a time, while relying heavily on visual feedback. As they become more and more proficient, they might remember the shape of the strokes corresponding to a particular dialog box and

rely less and less on visual feedback. While menu systems such as the Marking Menu were designed to encourage such transitions in the case of single command selections, we believe that this work is the first to explore how the same effect can be obtained for a succession of commands. Although it is certainly too early to judge the success of this approach at this point, our initial experience seems to imply that the natural use of the pen in an interaction setting with the computer strongly supports such compositions. For example our implementation shows how crossing might alleviate the need for dwell time for several interactions.

Somewhat like the keying system proposed by Zhai [24], we envision a system in which as novice users discover the interface, they also train themselves towards generating accurate gestures for the most commonly used commands. At some point, users will be able to remember the shape of the gesture well enough that it can be generated on top of the dialog box without the need for visual feedback. We believe that such a system could be implemented by having two concurrent tracking mechanisms for the user input. The first will be based on the system described above and will follow the crossing of each interface element. This mechanism will probably require visual feedback. The second tracking mechanism will track the user input and use a gesture recognition engine to classify the user input into possible strings of commands. Depending on aspects such as the start of the stroke, the scale of the stroke, or the overall speed, the input of both systems can be integrated to infer the user's commands.

Our implementation of the directory navigation system is the first step in that direction and shows how relaxing the strict constraints of goal crossing might help improve interaction fluidity. As shown in Figure 10, our first design for the directory navigator was based on a simple but rigid paradigm: the user will be building the prefix one letter at a time, from left to right by crossing a virtual crossbar with A at the top and Z at the bottom. While very simple in principle, this approach proved to be very difficult to manipulate. The layout creates abrupt changes in direction which causes the user to overshoot the path they are supposed to follow. By providing only one selection and letting the user create a crossing mark at its current location our current implementation provides a very similar conceptual model but simplifies the general interaction constraints on the user.

Space and time efficiency

In our experience, if one considers novice users, the space requirement of a crossing interface will be similar to the equivalent point-and-click interface. This is derived from the fact that crossing is as efficient as aiming, so one can simply substitute every standard button with a crossing button of the same size.

Yet, when one wishes to leverage command composition, a space speed trade-off will appear because some space will

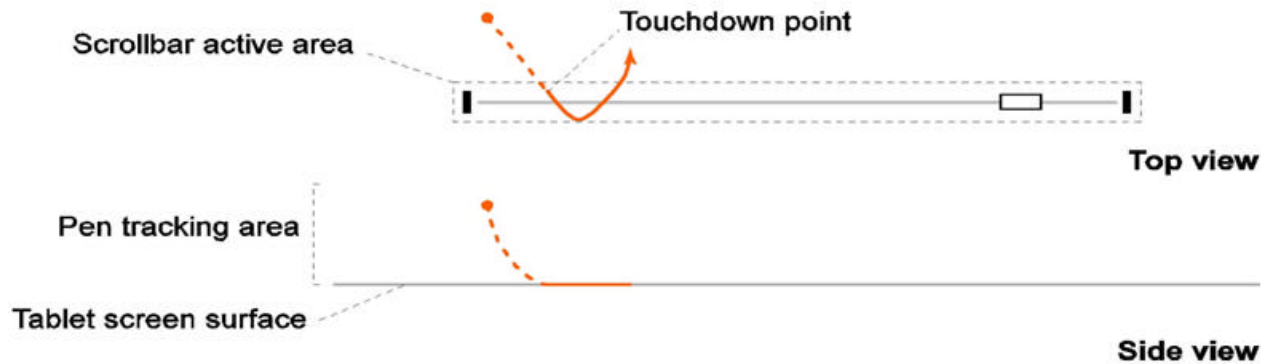


Figure 11 Leveraging the tracking information during a scrollbar interaction. If the system only uses the information gathered after the pen touches the screen (shown here as a solid line), it may be difficult to recognize the intended gesture since the first stroke is very small. Taking into account the information gathered while the pen is in tracking range (shown here as a dotted line) can greatly improve gesture recognition since the system can observe a longer stroke.

be needed due to the sloppiness of rapid gestures. From our experience, we believe that a slightly larger footprint might prove to be acceptable as the expected speed benefits from command composition are substantial. Furthermore, natural constraints of efficient visual layout (such as the use of negative space as described in Mullet et al.[14]), might prove to be all that is needed. Of course, it is too soon to know for sure and we intend to conduct user experiments to confirm or disconfirm this conjecture.

Navigation through large lists (or hierarchies)

Our exploration of the crossing interface led us to a novel way to navigate large lists (and by extension large hierarchies) which seems more efficient and fluid than the traditional list box approach. This problem has been explored before using speed dependent zooming [11], geometric Fisheye distortion in the FishEye menu [5] and user's directed pruning of the hierarchy as in Favorite Folders [12]. By using the prefix hierarchy as the basis for our progressive disclosure strategy (a fisheye in the general sense described by Furnas [6]), we create the pen equivalent of the keyboard based auto-completion system. This approach (somewhat similar to the Dasher [22] predictive text entry mechanism) limits the number of choices to be performed by users and offers a more fluid way to navigate hierarchies.

While we demonstrated this system for lists, it can be applied to any data set for which one can define the notion of an ordered prefix hierarchy. This includes information such as date (structured by year, month, day, hour...), but also any tabular data with columns which have a natural order.

Hardware and software considerations

Tracking limitations

Early Tablet-PCs were unable to track the pen outside the screen area. This causes problems when a gesture is started on the screen but extended outside of it. This problem is common in the direct setting, and could be easily addressed

by extending the tracking area beyond the limit of the screen. Note that while newer models, such as the Toshiba Portégé are doing just that, the mouse information provided to the application framework is still clipped at the boundary of the screen. We believe that providing the pen coordinates outside the boundary of the screen (at least for requesting applications), will significantly improve the usability of these devices.

Another problem we observed was that sometimes users start their crossing gesture before landing the pen and land the pen very close to the crossing threshold (Figure 11). While this is not a problem for simple widgets such as buttons, it makes it difficult to recognize the intended gesture before the line is crossed and feedback needs to be provided (this might happen for example when setting the absolute position of a document). To address this problem, our system keeps a small queue of pen positions when the pen is flying over the tablet. Values in that queue are used at pen touchdown as a way to prime the gesture recognition and increase its reliability.

Other devices

While our interface was developed for pens on a Tablet-PC, the results presented here can also be applied to other configurations, such as digital whiteboards and desktop computers using either a traditional mouse or a pen.

FUTURE WORK

We are planning to develop a richer toolkit of widgets to extend the scope of this study. We would also like to develop a set of design rules which help to design applications based on crossing. As part of this effort, we are planning on an extensive user evaluation program to investigate both low level interactions (such as crossing a single goal) as well as the compound commands.

Beyond visual feedback

We are also investigating ways to foster a rapid transition from visually-oriented interaction to gesture based interaction. Our current prototype is already using sound in

some cases (e.g. during the directory navigation). Tactile feedback transmitted from the screen through the pen tip seems another obvious candidate. We are planning to explore how new haptic techniques simulate the feel of physical buttons on displays [16, 19] could be extended to create “haptic channels”. This might help users to navigate through complex dialog boxes with minimum visual feedback.

CONCLUSION

We presented the first exploration of crossing as the primary building block of a graphic user interface. We found that crossing is as expressive as the more traditional point-and-click and provides designers with more flexibility than the other because it takes into account the shape and direction of the strokes. We also found that a crossing interface can encourage the fluid composition of commands in one stroke and illustrated this feature with several examples such as our find and replace window. We believe that this fluid composition of commands might lead to more efficient and natural interfaces for pen-based computing. We also believe that this finding might be applied in other domains such as whiteboard environments and mouse-based desktop computing.

ACKNOWLEDGEMENTS

The authors would like to thank Grecia Lapizco-Encinas and Alejandro Rodriguez who implemented an early prototype of a crossing interface as part of a graduate HCI seminar project. We also wish to thank Corinna Löckenhoff, Anja Szustak and all other reviewers of early drafts of this paper for their comments. This work has been supported in part by Microsoft Research for the Microsoft Center for Interaction Design and Visualization at the University of Maryland. Dave Levin was drawing the screen content in Figure 1.

REFERENCES

1. Accot, J. and S. Zhai. Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. *Proceedings of CHI'97*, pp. 295 - 302.
2. Accot, J., Les Tâches Trajectorielles en Interaction Homme-Machine—Cas des tâches de navigation., PhD thesis, Université de Toulouse 1. 2001
3. Accot, J. and S. Zhai. More than dotting the i's --- foundations for crossing-based interfaces. *Proceedings of CHI'03*, pp. 73 - 80.
4. Baudisch, P. Don't click, paint! Using toggle maps to manipulate sets of toggle switches. *Proceedings of UIST'98*, pp. 65 - 66.
5. Bederson, B.B. Fisheye menus. *Proceedings of UIST'00*, pp. 217 - 225.
6. Furnas, G.W. Generalized fisheye views. *Proceedings of CHI'86*, pp. 16 - 23.
7. Guimbretière, F. and T. Winograd. FlowMenu: combining command, text, and data entry. *Proceedings of UIST'00*, pp. 213 - 216.
8. Guimbretière, F., M. Stone, and T. Winograd. Fluid interaction with high-resolution wall-size displays. *Proceedings of UIST'01*, pp. 21 - 30.
9. Hong, J.I. and J.A. Landay. SATIN: a toolkit for informal ink-based applications. *Proceedings of UIST'00*, pp. 63 - 72.
10. Hopkins, D., The Design and Implementation of Pie - Menus. *Dr. Dobb's Journal*, 1991. **16**(12): p. 16 - 26.
11. Kurtenbach, G., The design and Evaluation of Marking Menus, PhD thesis, University of Toronto. 1993
12. Lee, B. and B. Bederson, Favorite Folders: A Configurable, Scalable File Browser. 2003, UMD.
13. Masui, T., K. Kashiwagi, and I. George R. Borden. Elastic graphical interfaces to precise data manipulation. *Proceedings of CHI'95*, pp. 143 - 144.
14. Mullet, K. and D. Sano, *Designing Visual Interfaces: Communication Oriented Techniques*. 1994: Prentice Hall.
15. Mynatt, E.D., T. Igarashi, W.K. Edwards, and A. LaMarca. Flatland: new dimensions in office whiteboards. *Proceedings of CHI'99*, pp. 346 - 353.
16. Nashel, A. and S. Razzaque. Tactile virtual buttons for mobile devices. *Proceedings of CHI'03*, pp. 854 - 855.
17. Pederson, E.R., K. McCall, T.P. Moran, and F.G. Halas, Tivoli: an electronic whiteboard for informal workgroup meetings, in *Human Factors in Computing Systems. INTERCHI '93*. 1993, IOS Press: Amsterdam, Netherlands. p. 391-8.
18. Pook, S., E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus: execution and control in a single interactor. *Proceedings of CHI'00 Extended Abstracts*, pp. 263 - 264.
19. Poupyrev, I. and S. Maruyama. Tactile interfaces for small touch screens. *Proceedings of UIST'03*, pp. 217 - 220.
20. Ramos, G. and R. Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. *Proceedings of UIST'03*, pp. 105 - 114.
21. Saund, E., D. Fleet, D. Larner, and J. Mahoney. Perceptually-supported image editing of text and graphics. *Proceedings of UIST'03*, pp. 183 - 192.
22. Ward, D.J., A.F. Blackwell, and D.J.C. MacKay. Dasher—a data entry interface using continuous gestures and language models. *Proceedings of UIST'00*, pp. 129 - 137.
23. Winograd, T. and F. Guimbretière. Visual instruments for an interactive mural. *Proceedings of CHI'99 (Extended Abstracts)*, pp. 234 - 235.
24. Zhai, S. and P.-O. Kristensson. Shorthand writing on stylus keyboard. *Proceedings of CHI'03*, pp. 97 - 104.