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 developed as a benchmark to demonstrate the feasibility of 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 which supports the development of 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.

CATEGORIES AND SUBJECT DESCRIPTORS

H.5.2 Graphical User Interfaces, Input Devices and Strategies; D.2.2 User Interfaces; I.3.6 Interaction Techniques

ADDITIONAL KEYWORDS AND PHRASES

Crossing based interfaces, command composition, fluid interaction, pen-computing

INTRODUCTION

The recent introduction of portable, pen-based computers has demonstrated that, while very powerful, the standard WIMP-interface (Windows, Icons, Menus, and Pointers) 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.

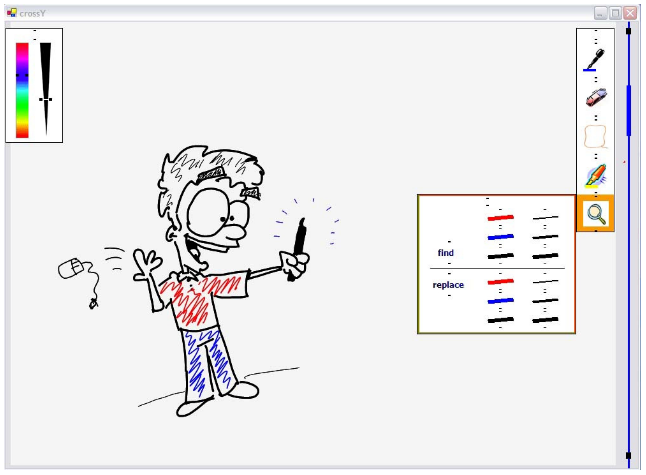


Figure 1 The CrossY interface showing the brush-palette (left) and the main palette with a find/replace dialogue box (right).

Several solutions have been proposed to address these problems. However, by its very nature, the design paradigm of current Graphical User Interfaces (GUI) is not well adapted to the pen's natural affordance of drawing strokes. Traditional point-and-click interfaces insist on segmenting user interactions in a sequence of point-and-click interactions. Using such interfaces with a pen may be frustrating, as users are forced to 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 can be 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 [13] and Baudish's toggle maps [5]), designers have not explored the potential of crossing as a building block for GUIs.

In this paper we begin to systematically explore 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 unique challenges of developing such a crossing-based 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 layout is required. This factor is less important in traditional interfaces.

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 arises from the mismatch between interface and interaction device: while the current interfaces were designed in an indirect pointing configuration with a stable pointer controller, tablet computing provides a direct setting with a pen, a "noisy" input device. We believe that the problem has an even deeper root: pen use encourages a fluid, continuous style of interactions based on strokes, whereas point-and-click interfaces insist on segmenting interactions into a series of pointing steps.

To address this fundamental issue, we decided to explore the use of crossing instead of pointing as suggested by Accot et al. [3]. CrossY, a simple drawing application, was developed to examine the strengths and weaknesses of crossing as a building block of interaction design.

We decided to limit the scope of this early exploration by focusing on the following key aspects:

- **Expressiveness.** One of the most important questions to be addressed is: *Can the new language express as rich a set of features as the language it means to replace?* Therefore, we decided to examine how the key elements of a basic WIMP interaction can be implemented in a crossing interface. As a starting point we decided to implement standard buttons, scrollbars, menu systems, dialog boxes (including selection of items from a list) and a simple set of window management tools (Figure 2). In each case, our initial goal was to mimic existing capabilities before developing new features.

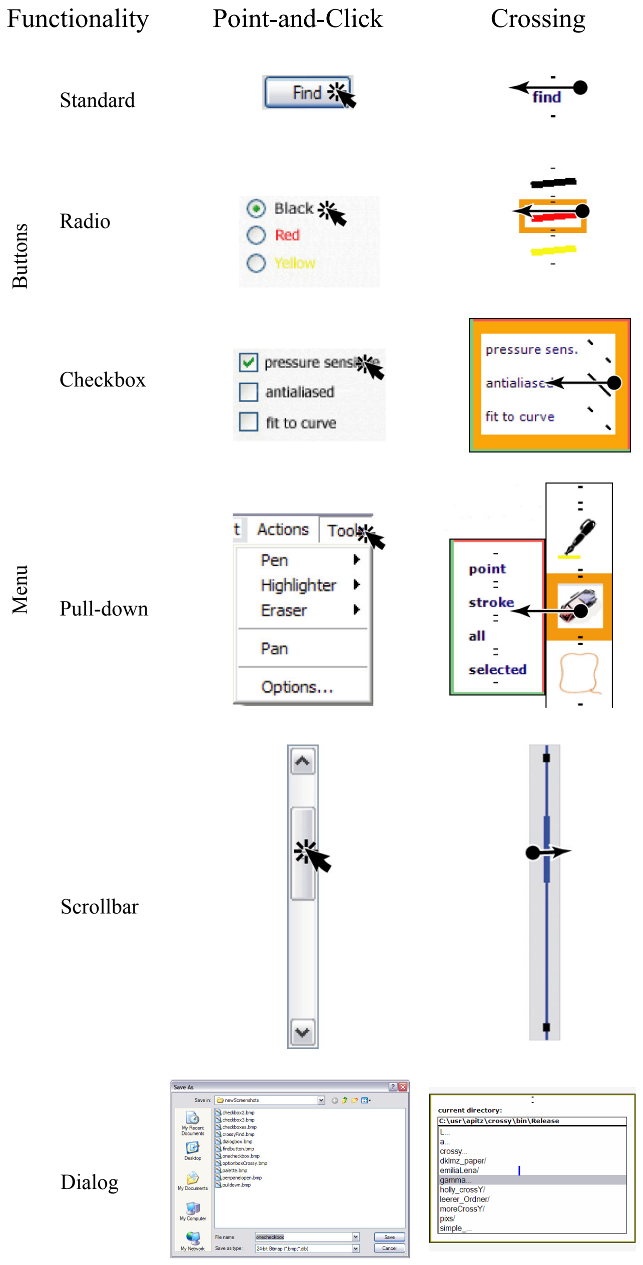


Figure 2 Correspondence table showing elements of traditional, point-and-click GUIs and their CrossY counterparts.

- **Fluid composition of commands.** As illustrated by Lotus Notes [13] and the toggle maps system [5] interfaces based on goal crossing promote the fluid composition of commands. This allows users to issue several actions (e.g., 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 advantages of transitioning from a visual interface to a gesture based interface (as demonstrated in the Marking Menu [15]) could be extended to the selection of several commands inside a dialog box.

- **Efficiency.** Expressiveness and fluidity are of little use if they come at the price of 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 needs to use it efficiently. Crossing-based interfaces are unique, since the visual layout affects both the composition and the efficiency of commands.

It is important to note that in contrast to previous conceptual explorations (such as Winograd and Guimbretière [29]), we did not focus on the creation of new, application-specific interactions. This is a deliberate choice. Focusing on standard widgets gives us a reference point against which our design can be evaluated.

PREVIOUS WORK

Several previous systems have departed from strictly point-and-click interfaces. One example is Lotus Notes [13] 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 [5] in which users can draw on top of a set of toggle buttons to trigger them in one gesture 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 [29]. 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 out 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.

Many pop-up menu systems are well adapted for pen-based interaction. Several systems, such as Pie Menu [12] and Marking Menu [15], use direction and pen-up transition to select commands. Other menu systems such as Control Menu [24] and FlowMenu [9] use crossing as a way to select commands. Since our system is crossing-based, we decided to use FlowMenu as our primary pop-up menu system.

In the recent years, several systems also challenged the use of the point-and-click interface for pen computing in whiteboard environments such as Tivoli [22], FlatLand [20] and PostBrainstorm [10], on the desktop [26], or for pen computing [27]. 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 systems, such as SATIN [11], have explored gesture-based interactions. Although gestures are important to crossing-based interfaces, the gestures implemented in CrossY are relatively simple and, by adding a crossing requirement, ambiguity is reduced. In that respect our system is similar to Geißler's Gedrics [8], a system in which users can select the action performed by an icon by drawing a given gesture on top of it.

CROSSY

CrossY is a simple sketching program offering several tools (e.g., a pen, a highlighter, an eraser). It was designed to run on the Tablet-PC platform without a keyboard. CrossY offers a simple search-and-replace feature which lets users find strokes based on their attributes (color and thickness) and replace them. It also let users modify tool attributes. Although this drawing system is primitive by today's standards, CrossY demonstrates how most of the standard widgets of point-and-click interfaces (Figure 2) can be implemented in a goal crossing framework.

Command selection

Like many drawing applications, the CrossY interface implements two kinds of menu systems. Common tools are accessed through a tool palette placed on the right of the display (see Figure 1). This layout was adopted to limit

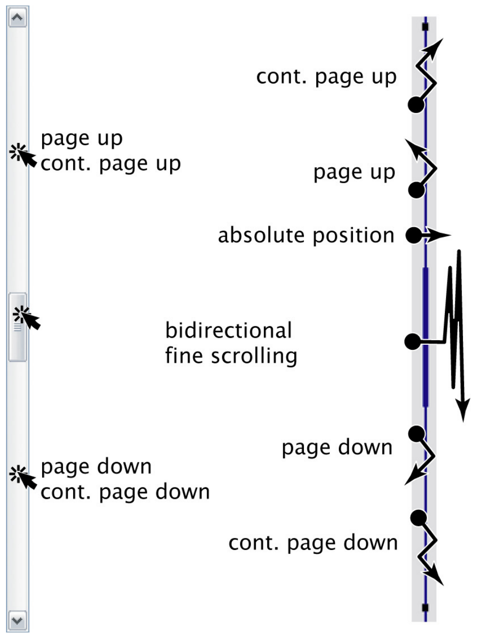


Figure 3 Comparison of the traditional scrollbar to our scrollbar. The stars indicate a click. The dots indicate that the pen is touched to the screen, and the strokes show the gesture which triggers the action.

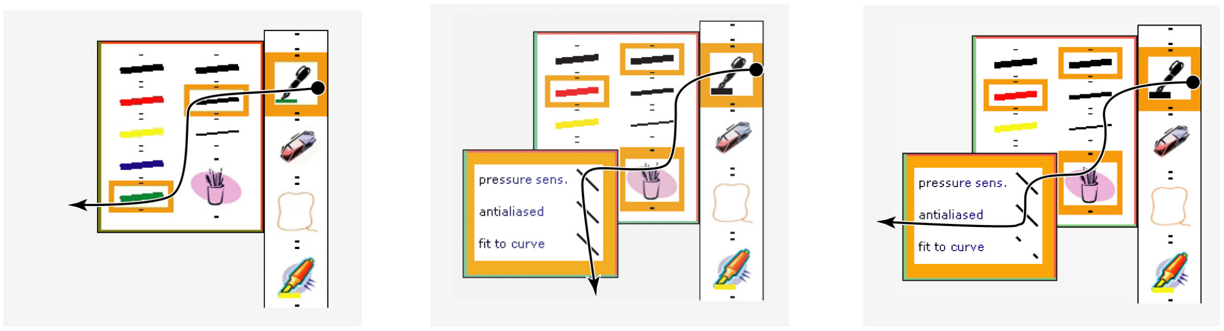


Figure 4 **Left**: The CrossY palette with the pen-panel opened. A single stroke opens the pen-panel, selects width and color of the strokes, and validates the selection. By convention, the left and bottom edges of each dialog box are validation edges (shown in green), and the top and right edges are cancellation edges (shown in red). **Middle**: The dialog box with check boxes to set the stroke-rendering attributes. A single stroke selects all items. **Right**: The dialog box with the check boxes to set the stroke-rendering attributes. A single stroke selects only two items.

potential hand occlusion. CrossY offers five basic tools to choose from: a pen, an eraser, a lasso, a highlighter and a search tool. Each of these tools can be selected by simply crossing its icon from right to left. Users can also move the palette to a more convenient place. To do so, users cross the center of the title bar between the two black marks from left to right. This action starts the dragging interaction which will stop as soon as the pen is lifted from the screen. Crossing the same area from right to left brings the palette back in its 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 3. The crossbar looks like a simple bar spanning the length of the document viewport and shows 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 3). To start a continuous page down or page up, the user simply crosses the bar a third time after issuing the initial command. The document now scrolls continuously until the pen is lifted.

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. 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 initial gain is defined by the ratio of the document length to the scrollbar length, the gain can be reduced during the adjustment phase to allow for finer adjustments. While some experimental scrollbars such as the FineSlider [17] provide similar options, the fluid integration of the two phases is typically difficult to achieve in a point-and-click interface. Another advantage of the crossbar is that users are not required to reach a given area of the bar before interacting. For example, they can initiate scrolling commands anywhere on the scrolling area. They also don't need to acquire the crossbar's slider before moving to an absolute position in the document; they just need to cross the crossbar at the target position. This makes the scrolling process faster and reduces the reliance on visual feedback.

Selecting pen attributes

In CrossY, users can select pen attributes by using either 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. Unlike current implementations, which present "dual-use" in a tool palette (such as in Adobe Illustrator [4]), 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 such 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 horizontally or vertically (Figure 4, middle). For one, this feature reinforces the fact that the check boxes are not mutually exclusive. Further, we noticed that it was difficult to cross only one item with a vertical stroke. 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

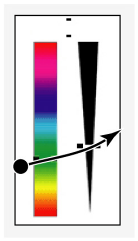


Figure 5 The brush-palette and a single stroke which selects color and width. The two small lines in the title bar shows the crossing position to move the widget.

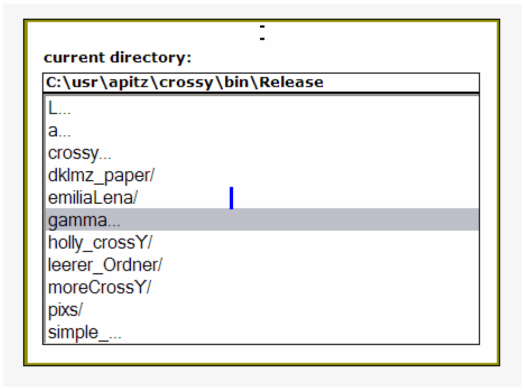
mechanism. The corresponding buttons are in fact very close to the edge of the window. Both the bottom and left border are validating borders (shown in green in our implementation), while the top and right border are cancellation borders (shown in red in our implementation). This layout lets users select all relevant 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 specific gesture (see Figure 5).

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 a dialog box which is structured around two panels (Figure 6). On the top panel, the user can select the width and color of the target strokes using a set of radio buttons. On the bottom panel, the user can select the new width and color 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. Similarly, replacement is triggered by crossing the "replace" button from left to right



current directory:

C:\usr\apitz\crossy\bin\Release

L...
a...
crossy...
dklmz_paper/
emiliaLena/
gamma...
holly_crossY/
leerer_Ordner/
moreCrossY/
pixs/
simple_...

Figure 7 The file dialog box. The small vertical line in the middle of the widget is the crossing target for expanding a prefix or opening a directory/file.

(Figure 6, left). While this layout seems somewhat unusual, it has been selected to encourage command composition. For example a user can in one single 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 6, 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 6, 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. An undo for replacements is achieved by crossing the "replace" button from right to left.

Loading an existing drawing

The file dialog box (Figure 7) is called up through FlowMenu. It lets users navigate the file system and load an existing drawing. At first glance, using crossing to navigate the file system hierarchy seems like a challenge since current interfaces rely heavily on the use of sequential point-and-click operations for this function. In traditional

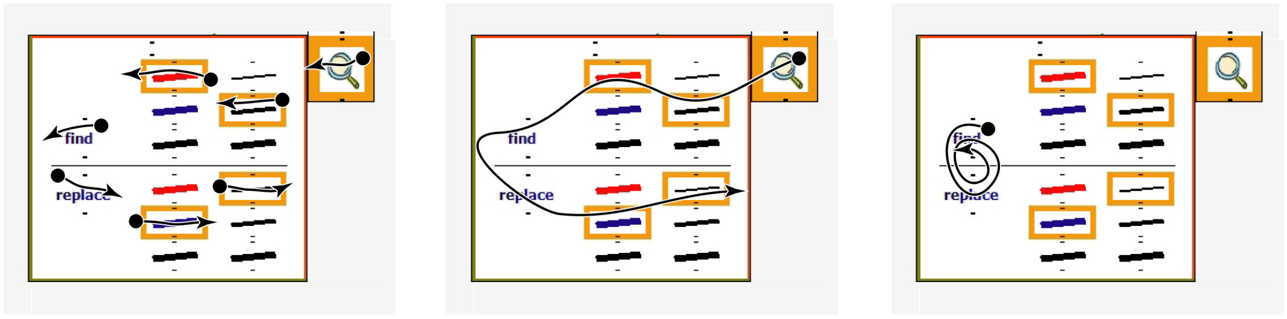


Figure 6 **Left**: Separate strokes are shown in the find-and-replace dialog box. Users select the values for the target stroke in the upper panel and the replacement values in the lower panel. **Middle**: The separate strokes are combined into one single stroke. **Right**: Repeated find-and-replace operations are carried out with one continuous stroke.

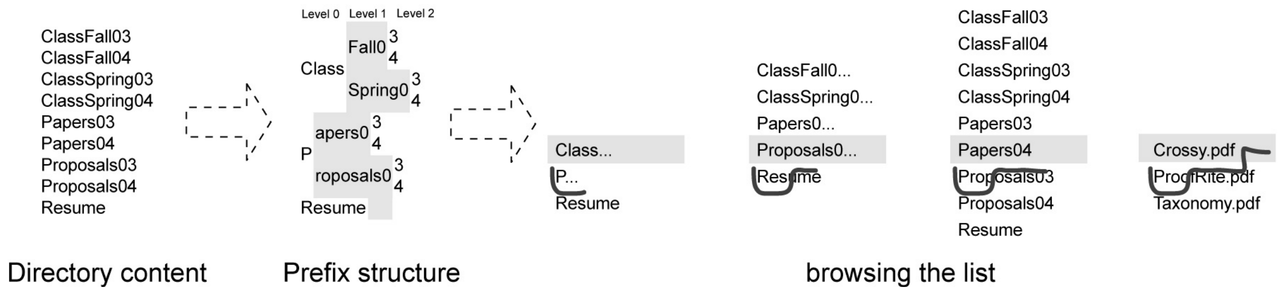


Figure 8 Exploring a directory. **Left**: The directory content. **Middle**: The corresponding prefix structure. **Right**: Navigating through a directory to open the file Papers04/Crossy.pdf. First, the prefix **P** is set in focus and expanded. Second, the prefix **Papers0...** is selected and expanded. Next, the **Papers04** directory is opened. Finally, **Crossy.pdf is** selected and opened.

navigation systems, users first have to search through the list of files that is present at the current level. This is typically achieved by using the scrollbar tab for coarse adjustment and the arrow at the end of the scrollbar for line by line movement. Next, users have to select the next directory (or the target file) 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 text based interface with auto-completion enabled. We believe that the crossing paradigm provides ways to combine the convenience of the graphical interface with the speed of auto-completion.

In our directory navigation tool, the local directory is scanned and its contents are parsed into a hierarchy of display levels. Exploration of the file hierarchy works on the same basis as auto-completion in text-based systems. 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. Exploration is performed by expansion of successive prefixes as users move through successive levels. 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, left/middle). It is important to note that there are only a limited set of possible characters (256 in theory but far less in practice) that may 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 8, right): At all times, the currently selected item is presented highlighted at the center of the widget. Users can change the currently selected item by moving the pen up and down anywhere on the widget. To move one level downward in the display hierarchy, users simply make a left-to-right horizontal movement in the current gesture. This causes the current highlighted prefix (represented with an ellipsis, e.g. "P…") to extend one level. A movement to the right while an unambiguous name is selected, loads the corresponding directory or file. To move one level upward in the display hierarchy, users need to make a small right-to-left horizontal movement in their gesture. 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 provide a "click" sound each time a transition between levels occurs 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 is very efficient to navigate through large directory structures given that the number of levels in the prefix structure of each directory is typically small. This allows the user to navigate through several directory levels in the space of a small window.

Implementation

Our system was implemented in C# on a Tablet-PC using the Windows XP Ink API and the .NET framework [14] as the basis for our design. However, it could be easily ported to any other language or operating system as it relies primarily on basic windowing constructs (with the exception of ink management).

DISCUSSION

CrossY was implemented as a platform to investigate how crossing may improve the overall fluidity of pen-based interactions 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 experiences gained while implementing CrossY it is clear that the crossing paradigm is at least as expressive as the standard point-and-click interface and provides the same level of functionality as the latter. It is possible to offer a wide range of features with a minimal visual footprint on the screen because the system accounts for the crossing location (in the crossbar and the palette for selecting pen attributes), the performed gesture (in the crossbar), and the direction of the stroke. Similar advantages were achieved in Gedrics [8] which uses gestures on top of icons. We now report on the insights we gathered while building our prototype.

Overloading versus easy discovery

Overloading different functions on top of the same visual artifact is certainly attractive from the designer's point of view. However, this approach raises the problem of discovery: users need to "learn" the system as not every interaction is self-explanatory in the first place. This is not a new problem in interface design and was identified in many systems such as the Marking Menu [15]. There are several techniques to facilitate discovery. First, compared to pure gesture based systems (such as [11]), the crossing system provides visual cues suggesting that some actions are 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. For example, this set includes the typical direction used to perform an action as well as movement in the reverse direction as a natural undo for the action. 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. We are also considering to implement the same technique as in Gedrics [8]: drawing a question mark gesture on top of the widget will present a description of the widget features.

Fluid composition of commands

Another interesting aspect of the crossing paradigm is the possible composition of commands in one single stroke. We see the feature of command composition as a unique and 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 start to remember the shape of the strokes corresponding to a particular dialog box and rely less and less on visual feedback. As described earlier, each command combination can also be executed in separate steps. This reduces the cost of making a mistake while performing a long sequence of actions because the user only needs to restart the gesture at the point where the error occurred. 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.

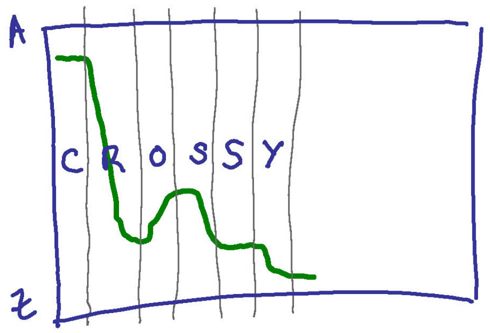


Figure 9 Original design for the directory navigation system. The system was based on an absolute mapping scheme (A on top and Z at the bottom) with a transition from one level to the next on strict boundaries, shown as light lines.

Although it is certainly too early to judge the success of the composition approach, our initial experience implies that the natural use of the pen in an interaction setting with the computer supports command compositions. For example, our implementation demonstrates how crossing may alleviate the need for dwell time for several types of interactions.

Somewhat like the keying system proposed by Zhai [30] (and Quikwriting [23]), 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 to be able to generate it on top of the interface elements without the need for visual feedback. We believe that such a system could be implemented by having two concurrent tracking mechanisms for user input. The first mechanism will be based on the system described above and will track the crossing of each interface element. This mechanism will typically require visual feedback. The second tracking mechanism will use a gesture recognition engine to classify user input into possible strings of commands. Depending on specific aspects such as the start of the stroke, the scale of the stroke, or its 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 can help to improve interaction fluidity. As shown in Figure 9, our first design for the directory navigator was based on a simple but rigid paradigm: the user had to build 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

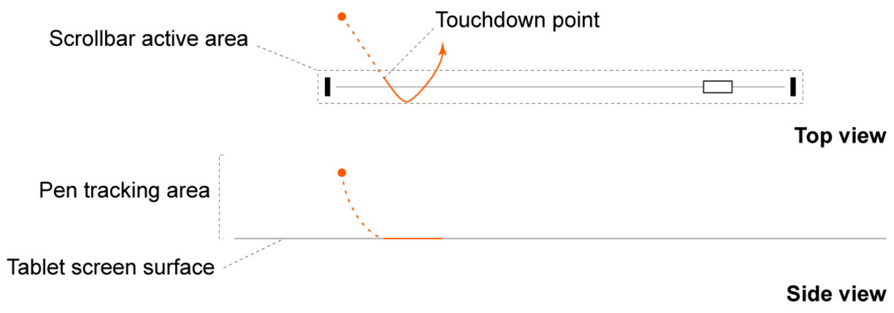


Figure 10 Leveraging the tracking information during a scrollbar interaction. If the system only uses the information gathered after the pen touches the screen (shown here as a solid line), it may be difficult to recognize the intended gesture since the first stroke is very small. Taking into account 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.

which causes users 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-based interface will be similar to the equivalent point-and-click interface. This is derived from the fact that the crossing efficiency is similar to that of aiming [3], 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 vs. speed trade-off will appear because some space will be needed due to the sloppiness of rapid gestures. This means that we need to provide more space for each widget in order to ensure reliability. Based on our experience, we believe that a slightly larger footprint may 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. [19]), may 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-based 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 in speed-dependent zooming [11], geometric Fisheye distortion in the FishEye menu [6], user's directed pruning of the hierarchy as in Favorite Folders [16], and in a combination of data visualization, keyword and category search in Masui’s multi-view information retrieval system [18]. By using the prefix hierarchy as the basis for our progressive disclosure strategy (a fisheye in the general sense described by Furnas [7]), we create the pen equivalent of the keyboard based auto-completion system. This approach (somewhat similar to the Dasher [28] 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 an ordered prefix hierarchy. This includes information such as date (structured by year, month, day, hour…), but also any tabular data with columns that have a natural order.

Consistency

In general the consistency over the whole application is stringent in the sense that crossing widgets from right to left triggers the action. This creates the main interaction direction (right to left) which minimizes occlusion for right-handed users. There are nevertheless several exceptions to this rule in our system. The first concerns the find-and-replace dialog box. In this case, the replace button is crossed from left to right in order to trigger the replacement. This enables the user to control the whole find-and-replace-dialog with one single stroke, and makes it possible to use the reverse movement as “undo replace” for the replace button and as “find backwards” for the find button. This potential for command composition was judged to be more important that consistency. The second instance concerns the file dialog box. Here, the role of the normal reading direction in Western languages was so overwhelming that it seemed more important than consistency. For both cases, it is not clear yet, if these inconsistencies are a source of confusion for users. Note also that sometimes we offer more flexibility. For example, dialog boxes have two validation borders (left and bottom) and two cancellation borders (right and top). This was required to simplify “exit” paths.

Hardware and software considerations

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. 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.

We further observed that sometimes users started their crossing gestures before landing the pen. Also, they often landed the pen very close to the crossing threshold (Figure 10). 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. For example, this might happen 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.

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.

Initial user feedback

While we have not yet conducted formal user testing, we can report on informal user feedback gathered during a public demonstration of CrossY during our lab's open house. Generally, reactions were very positive and all users liked the basic idea and interaction style. Yet, we observed that it was difficult for naïve users to discover how to use the interface without some initial explanation. Once the system was briefly demonstrated, they adapted the new technique very rapidly and were able to use it without any further assistance. None of the users considered initial problems with the system as a fundamental hindrance and the overall opinion is best described by the comment of one user: "We are so used to point-and-click, it will take a while to get used to crossing." Several of the users worked with CrossY in direct comparison to standard drawing applications and all users agreed that the way of interaction with CrossY was much more intuitive and better suited for the task (i.e., drawing) and the tool (i.e., a pen).

FUTURE WORK

We are planning to develop a richer toolkit of widgets to extend the scope of our work. This toolkit will provide the standard widgets used for building graphical interfaces as well as new widgets that emerge from the new interaction technique. 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 an extensive user evaluation program to investigate both low level interactions (such as crossing a single goal), and sound design rules. We would also like to investigate how to improve self-discovery.

Beyond visual feedback

Further, we are 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 that simulate the feel of physical buttons on displays [21, 25] 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 interface and provides designers with more flexibility than the latter because it takes into account the shape and direction of the strokes. We also found that a crossing-based interface can encourage the fluid composition of commands in one stroke. We illustrated this feature with several examples such as our find-and-replace dialog box. We believe that this fluid composition of commands will ultimately lead to more efficient and natural interfaces for pen computing. We also believe that our findings can 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-based 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 drew the screen content of Figure 1.

REFERENCES

1. Accot, J. and S. Zhai. Beyond Fitts' Law: Models for Trajectory-Based HCI Tasks. *Proceedings of Human Factors in Computing Systems, 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 Human Factors in Computing Systems, CHI'03*, pp. 73 - 80.
4. Adobe System Incorporated, Illustrator 10.
5. Baudisch, P. Don't click, paint! Using toggle maps to manipulate sets of toggle switches. *Proceedings of*

User Interface Software and Technology, UIST'98, pp. 65 - 66.

6. Bederson, B.B. Fisheye menus. *Proceedings of User Interface Software and Technology, UIST'00*, pp. 217 - 225.
7. Furnas, G.W. Generalized fisheye views. *Proceedings of Human Factors in Computing Systems, CHI'86*, pp. 16 - 23.
8. Geissler, J. Gedrics: the next generation of icons. *Proceedings of International Conference on Human-Computer Interaction, INTERACT'95*, pp. 73 - 78.
9. Guimbretière, F. and T. Winograd. FlowMenu: combining command, text, and data entry. *Proceedings of User Interface Software and Technology, UIST'00*, pp. 213 - 216.
10. Guimbretière, F., M. Stone, and T. Winograd. Fluid interaction with high-resolution wall-size displays. *Proceedings of User Interface Software and Technology, UIST'01*, pp. 21 - 30.
11. Hong, J.I. and J.A. Landay. SATIN: a toolkit for informal ink-based applications. *Proceedings of User Interface Software and Technology, UIST'00*, pp. 63 - 72.
12. Hopkins, D., The Design and Implementation of Pie-Menus. Dr. Dobb's Journal, 1991. **16**(12): p. 16 - 26.
13. IBM, Lotus Notes (http://www.lotus.com). 2004.
14. Jarett, R. and P. Su*, Building Tablet PC Applications*. 2002: Microsoft Press.
15. Kurtenbach, G., The design and Evaluation of Marking Menus, PhD thesis, University of Toronto. 1993
16. Lee, B. and B. Bederson, Favorite Folders: A Configurable, Scalable File Browser UMD,
17. Masui, T., K. Kashiwagi, and I. George R. Borden. Elastic graphical interfaces to precise data manipulation. *Proceedings of Human Factors in Computing Systems, CHI'95*, pp. 143 - 144.
18. Masui, T., M. Minakuchi, I. George R. Borden, and K. Kashiwagi. Multiple-view approach for smooth information retrieval. *Proceedings of User Interface Software and Technology, UIST'95*, pp. 199 - 206.
19. Mullet, K. and D. Sano*, Designing Visual Interfaces: Communication Oriented Techniques*. 1994: Prentice Hall.
20. Mynatt, E.D., T. Igarashi, W.K. Edwards, and A. LaMarca. Flatland: new dimensions in office whiteboards. *Proceedings of Human Factors in Computing Systems, CHI'99*, pp. 346 - 353.
21. Nashel, A. and S. Razzaque. Tactile virtual buttons for mobile devices. *Proceedings of Human Factors in Computing Systems, CHI'03*, pp. 854 - 855.
22. 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.
23. Perlin, K. Quikwriting: continuous stylus-based text entry. *Proceedings of User Interface Software and Technology, UIST'98*, pp. 215 - 216.
24. Pook, S., E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus: excecution and control in a single interactor. *Proceedings of Human Factors in Computing Systems, CHI'00 Extended Abstracts*, pp. 263 - 264.
25. Poupyrev, I. and S. Maruyama. Tactile interfaces for small touch screens. *Proceedings of User Interface Software and Technology, UIST'03*, pp. 217 - 220.
26. Ramos, G. and R. Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. *Proceedings of User Interface Software and Technology, UIST'03*, pp. 105 - 114.
27. Saund, E., D. Fleet, D. Larner, and J. Mahoney. Perceptually-supported image editing of text and graphics. *Proceedings of User Interface Software and Technology, UIST'03*, pp. 183 - 192.
28. Ward, D.J., A.F. Blackwell, and D.J.C. MacKay. Dasher—a data entry interface using continuous gestures and language models. *Proceedings of User Interface Software and Technology, UIST'00*, pp. 129 - 137.
29. Winograd, T. and F. Guimbretière. Visual instruments for an interactive mural. *Proceedings of Human Factors in Computing Systems, CHI'99 (Extended Abstracts)*, pp. 234 - 235.
30. Zhai, S. and P.-O. Kristensson. Shorthand writing on stylus keyboard. *Proceedings of Human Factors in Computing Systems, CHI'03*, pp. 97 - 104.