HoverCross: A New Selection Paradigm for Pen-Based Interfaces

Alice Zhu
Harvey Mudd College
Claremont, CA
xzhu@hmc.edu

Scott Parkey
Harvey Mudd College
Claremont, CA
sparkey@hmc.edu

Christine Alvarado
Harvey Mudd College
Claremont, CA
alvarado@cs.hmc.edu

ABSTRACT

A central problem in pen-based interfaces is how to transition smoothly between drawing and editing. Separate drawing and editing modes can be awkward and distracting, while modeless editing gestures are error-prone. We present HoverCross, a seamless inking and editing interface that provides a simple and reliable method for users to select on-screen objects. HoverCross combines the strengths of several recent developments in pen-based interfaces. With HoverCross, users ink normally and then select objects or ink strokes by crossing over them in the hover space above the tablet screen. They can then edit their selection through a context menu on the canvas. User feedback indicates that HoverCross provides an efficient, fluid and robust transition between drawing and editing. Furthermore, most users prefer HoverCross over existing interfaces for several common diagram creation and editing tasks.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Hover space, pen input, tablets.

INTRODUCTION

The promise of pen-based interfaces is that they provide a natural way for users to create diagrams and take free-form notes. However, the integration of diagram creation and diagram editing remains a barrier to the widespread use of these interfaces. Pens are more convenient and natural for drawing, but editing with a pen remains cumbersome when the user is forced to rely on graphical user interfaces designed for a mouse and keyboard. Because the user must use the pen for both drawing and editing (or suffer the inconvenience of switching between the pen and the keyboard), a core challenge for pen-based computing is to construct an interface that allows users to switch easily between the two tasks, while allowing the system to unambiguously interpret a given pen stroke as drawing or editing.

Many proposed solutions to this problem have been brought in recent years. Traditional solutions (e.g., Windows Journal) require the user to enter "edit mode," usually by pressing a software button. This solution is simple, but the problems with modes are well known [9]. Using hardware buttons (e.g., the bezel buttons on the side of the Tablet PC) to trigger mode switching helps solve these issues because the mode switch is temporary—as soon as the user releases the button, the interface reverts to drawing mode. Thus, there is little potential for mode confusion. Studies suggest that this approach can be quite effective and natural [5, 3]. However, simply pressing the button requires not only extra physical effort, but in many cases also an extra hand.

Other researchers take a recognition-based approach [7, 10], attempting to distinguish automatically between drawing and editing strokes (e.g., lasso selections or gestures). However, even with choice mediators [6] to help resolve recognition ambiguities, recognition errors can be confusing. Furthermore, as we argue below, it is not clear that lasso selection is optimal for many selection tasks.

Recently, researchers have explored using the hover space[1] to invoke editing commands. Grossman et al. present Hover Widgets [2], in which the user performs gestures in the hover space to activate editing menus. Following on this work, Subramanian et al. explore the possibility of using several layers of the hover space to perform different editing tasks [8], while Kattinakere et al. formally model users' ability to track (e.g., execute gestures) in hover space [4].

While each of these recently developed approaches brings us closer to the dream of seamless pen-based drawing and editing, we believe our arsenal of drawing and editing techniques is not yet complete. Specifically, for common diagram creation and editing tasks, Hover Widgets or a multi-level hover space solution may be too heavyweight.

We explore the power and simplicity that can be obtained by combining the simplest aspects of many existing techniques. Our interface, called HoverCross, combines the strengths of hover space editing and crossing-based selection (e.g., [1]), resulting in an interface that is easy to learn, reliable, and fast for common drawing and editing tasks. Furthermore, our approach integrates seamlessly with almost any other pen-based editing technique, so in the worst case users can simply

[1]The *hover space* is the space above the surface of a digital tablet where the pen is still tracked but does not generate ink or mouse events.

ignore it and fall back on traditional editing methods.

INTERACTION USING HOVERCROSS

In many modal pen-based interfaces (e.g., Windows Journal), it is *selection*, not editing in general, that is typically relegated to its own mode. This division makes sense, as selecting objects in a drawing is typically the first step in performing almost any editing task. Once the user selects an object, she can edit it via a menu or direct manipulation. An explicit selection mode seems necessary because any apparent selection stroke just as easily could be a drawing stroke.

The key insight behind HoverCross is that relegating the process of selection to the hover space allows users to switch seamlessly between drawing and selecting without pressing any buttons. Furthermore, because the user performs only selection in the hover space, a simple crossing interface suffices, and there is no need to perform gestures in the hover space. The system then leverages the context of the selection to give the user additional power through a gesture-invoked context menu or direct interaction with the selected objects.

In our interface, the user draws normally on the screen to create diagrams containing simple shapes, text, and ink. The diagrams and text the user draws are recognized by the Microsoft gesture and text recognition engines, while the unrecognized ink is divided into one selectable object per stroke.

To edit the diagram, the user hovers the pen briefly over the tablet to trigger selection. The brief pause prevents triggering selection every time the stylus enters the range, as it inevitably does on its way to the screen. The length of the pause therefore affects whether users accidentally enter into selection or must wait too long; half a second appears to work as a medium. Once the user triggers selection, a small vertical line, called the handle, appears in the middle of each object, anchored at the bottom (Figure 1). The vertical alignment keeps the selection gesture an intuitive horizontal motion. The user simply needs to cross the stylus over the handle in either direction in hover space to select the object. Once the object is selected, the handle realigns its anchor to the top of the shape, creating a switch-like feel, which helps prevent users from accidentally undoing their selection and keeps handles from becoming too obstructive. Crossing the realigned handle deselects the shape. Entering selection mode also causes small circles and squares to be drawn on the edge of shapes. Clicking and dragging these small shapes causes the shapes to be transformed: the square for resizing, and the circle for rotating. The user easily can cross over multiple objects to select all of them. To clear the selection, the user moves the pen in hover space off the edge of the screen.

Using the hover space for selection is particularly convenient for cluttered diagrams in which the user wishes to edit several objects separated by intervening shapes or text. The user can enter the hover space to cross one object and then immediately exit the hover space by moving the pen tip away from the screen. The selected object remains selected even when the pen is outside of hover space. To select another object, the user simply needs to re-enter hover space near that object. This ability helps keep intervening handles between

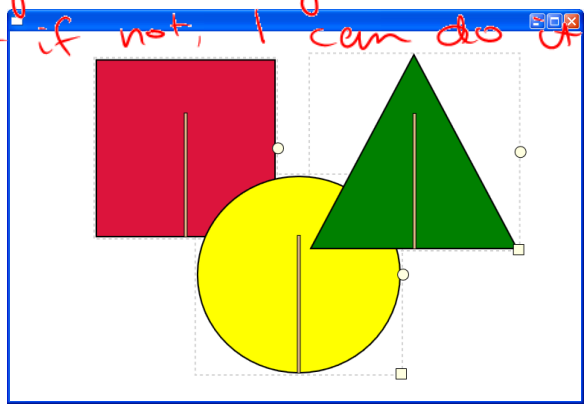

Figure 1: Vertical bars appear in each shape to give users a crossing target.

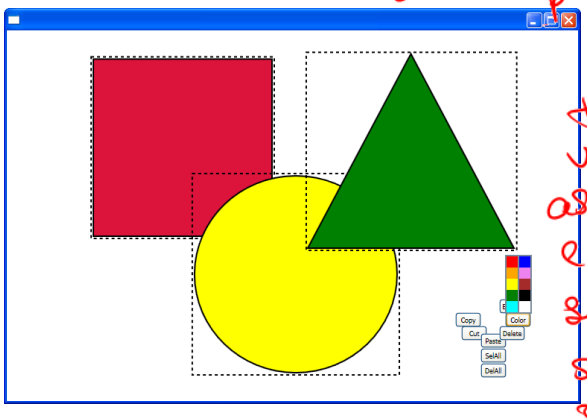

Figure 2: The context menu that appears when the user makes a simple gesture. The menu appears at the tip of the stylus at the end of the gesture, and actions apply to all selected shapes.

two desired objects from preventing the user from selecting both.

Once the user has selected at least one item, she can either edit the selected item, or continue to draw new portions of the diagram. The user has several options for editing the diagram. She can move selected shapes by dragging one of them with her pen on the screen. As long as she does not cross the handle in hover space, touching the shape will not deselect it. For more complicated editing tasks, the user can bring up a context menu around the tip of the stylus (Figure 2). We have altered the gesture for brining up the menu; currently, it requires the user to hold the stylus down, utilizing the straightforward and well-known Windows right-clicking gesture. The content of the menu depends on the type of objects selected, such as text or shapes. The user can navigate the menus by tapping the stylus on the menu options to quickly manipulate the selection. With little practice, the user easily can coordinate drawing, selecting, and editing in one fluid motion.

DESIGN DECISIONS

The HoverCross technique is appropriate for any application that combines pen-based drawing and editing. Our example application supports the creation of clean diagrams that

combine shapes and text, such as might be produced in PowerPoint or Visio. We chose this domain because creating polished diagrams typically requires more intensive editing than simply drawing freeform notes and diagrams. In a previous pilot study in which we observed people creating slides in PowerPoint, we found that they relied heavily on copying, pasting, resizing and moving shapes within their diagrams. This domain thus allows us to evaluate the utility of HoverCross for its intended purpose.

Confirming previous results [1], the crossing selection metaphor has several advantages over traditional lasso selection. For selecting single shapes or shapes roughly in a horizontal row, crossing the shapes' targets is much faster than lassoing them. Additionally, for shapes spread out in space, crossing in hover space behaves like "Control-clicking" with a mouse and keyboard, allowing users to select some objects while avoiding intervening ones.

A final strength of this interface is that any part of it may be implemented in conjunction with almost any other interface technique. If the user does not want to use it, he can simply ignore it. Because the hover selection interface requires a short pause to invoke, the user is not likely to trigger it by mistake, so it may be combined with a traditional modal selection interface. For example, a user might use modal lasso selection to select a group of tightly clumped objects, and then use HoverCross to deselect one of them. While we offer the context menu for convenience, additional menus may be added easily to the interface.

We conducted a small, preliminary user study to gain insight into how users compare HoverCross to existing selection methods. We compared three selection interfaces: an older version of HoverCross that utilized much of the existing functionality, including crossing selection; a variation of HoverCross called Cross; and the traditional lasso select. Cross differs from HoverCross in that users cross objects with the pen on the screen while holding down a nonpreferred hand button. Li et al. showed that using a nonpreferred hand button is the current most effective way to invoke edit mode [5]. Lasso select invoked using a modal interface button is the current approach in most commercial software. Seven users (3 female and 4 male) participated in our study. All were students at Harvey Mudd College, and all had experience using a tablet computer. No user had experience with a hover or crossing-based interface.

After receiving instructions and trying out the three interface techniques for about 2-5 minutes, users performed four tasks using each of the three interface techniques. We collected qualitative data on which interface users preferred for each task and quantitative data on the time it took for users to complete each task. Figure 3 shows task completion times. For tasks 1, 3, and 4, we found no significant difference in completion time between the three interfaces using a one-way within subjects ANOVA for each task. For these same tasks, four users preferred HoverCross over the other two interfaces (Figure 4).

We found that users enjoyed the transition between sketching and selecting, without the hassle of pressing a button or other

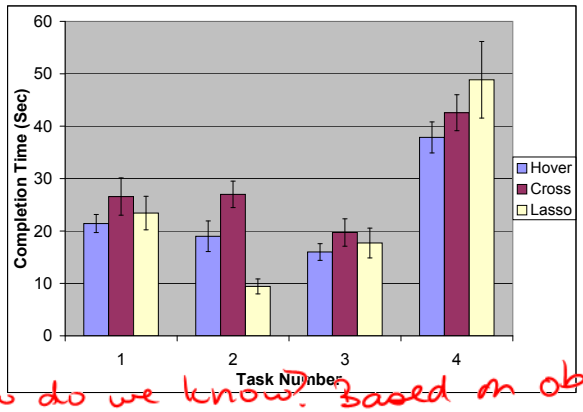

Figure 3: Average time to complete each task with each interface.

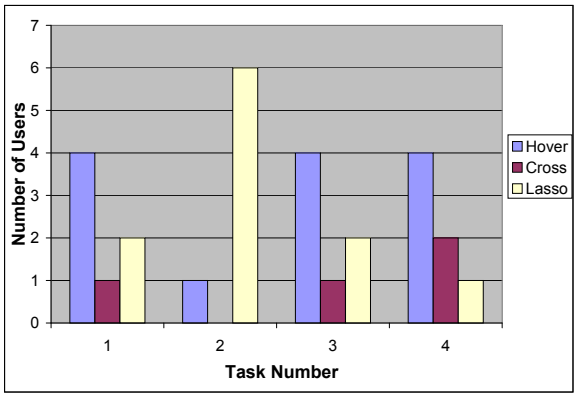

Figure 4: Number of users who preferred each interface for each task.

explicit indication, despite the reliability of the button. In addition, for small selections, drawing a circle around the objects seemed excessive. When objects were spread out over the canvas, they felt that selecting a few of them with lasso select was awkward, while with HoverCross they could simply move the pen across the screen to the objects of interest. This result suggests that there is a set of users that would benefit greatly from HoverCross in this situation.

For task 2, which involved selecting a group of closely placed objects, six of the users preferred lasso select. This result is understandable, because crossing over each individual object in the group is more cumbersome than drawing a lasso around it. We also found a significant difference between completion times. However, one user still preferred HoverCross for this task. He reflected that, while both interfaces are effective, HoverCross was more interesting and fun.

We altered several aspects of the program in response to the feedback we gained from the preliminary user study. First, we found that we needed to optimize the specific placement of crossing handles to deal with small objects and mostly occluded objects. Rather than having the selection handles fit the shapes precisely, we gave the handles minimum and max-

imum sizes, preventing the problem of having handles that are either too easy or too difficult to cross. We also changed the handles from static bars to switches that change position and color depending on whether their shape is selected to facilitate user interaction, and began to count crosses over obstructed handles as legitimate crosses, ensuring that all on-screen objects are selectable. Finally, to improve on the PowerPoint or Visio paradigm, we implemented resizing and rotation transformations for on-screen objects.

These improvements consistently tested well when given to users for feedback. Although allowing crossing selection of obstructed shapes was worrysome for the accidental selections it may produce, users rarely cited it as a problem, in contrast with the negative feedback we received about the inability to select obstructed shapes. Users also approved of the switch paradigm, and frequently commented that although it took some time to get used to, object selection became natural and fit well with the drawing system.

In these latter tests, users were told simply to draw a picture, some of which became fairly complex and included shapes overlapping other shapes. These pictures approximate the complexity of a typical diagram or PowerPoint slide, which will often include several overlapping shapes. The only complications came when shapes were grouped directly on top of other shapes. If, for example, one handle became grouped directly on top of another handle, both would be switched simultaneously when they were passed over. Although annoying, this was never a critical flaw, because users could simply resort to another means of selection, such as tapping the rotation circle and moving the shape on top. It is also worth noting that lasso selection does not solve this problem, either, since lassoing would always select both shapes as well. Indeed, HoverCross is probably an improvement in precision over lasso selection for cluttered diagrams, since the selection process is precise to a single, visible line segment.

The need for improvements to HoverCross remains. Users occasionally complained about the selection boxes becoming too cluttered, particularly for diagrams with many small strokes; this could be addressed by grouping strokes. Although the selection mode is effectively in its final stages of implementation, the rest of the prototype could use improvements in text recognition, menu access, and graphical issues. In addition, before we can make any positive claims about the effectiveness of the selection mode when deployed into other environments, we must deploy it into other environments and conduct additional user studies.

IMPLEMENTATION

This interface, designed for the Tablet PC, is written in C# using Windows Presentation Foundation and .NET 3.0. We use the built-in gesture and text recognizers. We recognize movement in the hover space by handling the StylusInAirMove event, tracking the stylus position, and selecting or deselecting an object whenever the stylus crosses its handle.

Crossing is determined by storing the previous stylus location and passing it, along with the current stylus location, to an intersection test. The intersection test checks whether the line segment between the two points crosses the center of any handles. If it does, the method returns all the relevant handles.

Our context menu is a collection of Button UIElements stored on the InkCanvas, each representing an editing option. When the user clicks on any of the buttons, the system performs the desired option on all selected items.

CONCLUSION

HoverCross combines many simple and effective ideas from many recent advances in pen-based interfaces to provide an elegant interface for inking and editing. Used in combination with other pen-based interaction techniques, HoverCross will bring us one step closer to the goal of creating pen-based interfaces that combine the freedom of paper with the power of the computer in a useful, and usable, way.

ACKNOWLEDGEMENTS

We would like to thank our users who helped us evaluate HoverCross. Much of this work was funded by the Baker Foundation and an NSF CAREER award (IIS-0546809).

REFERENCES

1. G. Apitz and F. Guimbretière. CrossY: A crossing-based drawing application. In *Proc. of UIST*, 2004.
2. T. Grossman, K. Hinkley, P. Baudisch, M. Agrawala, and R. Balakrishnan. Hover widgets: Using the tracking state to extend capabilities of pen-operated devices. In *Proc. of CHI*, 2006.
3. K. Hinckley, F. Guimbretière, P. Baudisch, R. Sarin, M. Agrawala, and E. Cutrell. The springboard: Multiple modes in one spring-loaded control. In *Proc. of CHI*, 2006.
4. R. S. Kattinakere, T. Grossman, and S. Subramanian. Modeling steering within above-the-surface interaction layers. In *Proc. of CHI*, 2007.
5. Y. Li, K. Hinckley, Z. Guan, and J. Landay. An experimental analysis of mode switching techniques in pen-based user interfaces. In *Proceedings of CHI*, 2005.
6. J. Mankoff, S. E. Hudson, and G. D. Abowd. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proc. of CHI*, 2000.
7. E. Saund and E. Lank. Stylus input and editing without prior selection of mode. In *Proc. of UIST*, 2003.
8. S. Subramanian, D. Aliakseyeu, and A. Lucero. Multi-layer interaction for digital tables. In *Proc. of UIST*, 2006.
9. L. Tesler. The smalltalk environment. *Byte*, 6:90–147, August 1981.
10. R. Zeleznik and T. Miller. Fluid inking: augmenting the medium of free-form inking with gestures. In *Proc. of Graphics Interface*, 2006.