HoverCross: A New Selection Paradigm for Pen-Based Interfaces

Author Name removed for blind review
Affiliation removed for blind review
Address removed for blind review
Email removed for blind review

Author Name removed for blind review
Affiliation removed for blind review
Address removed for blind review
Email removed for blind review

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 both directly and through an editing menu. User feedback indicates that HoverCross provides a promising new pen-based selection technique that offers an efficient, fluid and predictable transition between drawing and editing.

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 people have proposed solutions to this problem 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 [10]. Using hardware buttons 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 [6, 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 [8, 11], attempting to distinguish automatically between drawing and editing strokes (e.g., lasso selections or gestures). However, even with choice mediators [7] to help resolve recognition ambiguities, recognition errors can be confusing, and gestures can be hard to learn.

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 [9], while Kattinakere *et al.* formally model users' ability to track (e.g., execute gestures) in hover space [5].

While each of these recently developed approaches brings us closer to the goal 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 straightforward to learn, reliable, and convenient for common drawing and editing tasks. We do not anticipate that HoverCross will replace all other forms of editing (e.g. gestures or temporary mode-switching). Rather, our approach integrates seamlessly with most other pen-based editing techniques providing an alternative interaction style for users who prefer it.

[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.

INTERACTION USING HOVERCROSS

In many modal pen-based interfaces (e.g., Windows Journal), it is *selection*, not editing in general, that typically operates in 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 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 menu or direct interaction with the selected objects.

In our interface, the user draws normally on the screen to create diagrams containing simple shapes and raw ink. The Microsoft gesture recognition engine recognizes the shapes the user draws, while the unrecognized ink becomes one selectable object per stroke. Recognition is not necessary for HoverCross, but we included it in our interface so that we could explore using HoverCross to interact with both shapes and raw ink.

To edit the diagram, the user hovers the pen briefly over the tablet to trigger the handles for selection and simple editing tasks (Figure 1). The small vertical line that appears in the middle of the object, anchored at the bottom, is the selection handle. The small square and circle on the right of each object are resize and rotate handles, respectively. All of the handles for a shape are positioned on a faint rectangle that outlines the shape to which they apply, to help users correlate handles to shapes. We discuss the functionality of each handle in more detail below. The brief pause before handles appear prevents triggering the handles every time the stylus enters the range, as it inevitably does on its way to the screen. The length of the pause affects whether users accidentally trigger the handles or must wait too long. In our tests we found that half a second appears to work well, but we also suggest that the pause length be a user controlable feature. Additionally, even if the handles appear by mistake, the user can simply ignore them and continue to sketch.

To select an object, the user crosses the stylus in hover space over the handle in either direction. We chose a vertical alignment for the handles so that the crossing motion mimics the familiar horizontal motion of writing text. Once an object is selected, the rectangle surrounding the object darkens, and the handle realigns its anchor to the top of the shape, creating a switch-like feel (Figure 2). This realignment helps prevent users from accidentally undoing their selection, keeps handles from becoming too obstructive, and provides an additional visual cue to indicate which objects are selected. Crossing the realigned handle in hover space (again in either direction) deselects the shape. As the user crosses more objects' handles, these objects are added to the selection. To clear the whole selection, the user moves the pen in hover space off the edge of the screen.

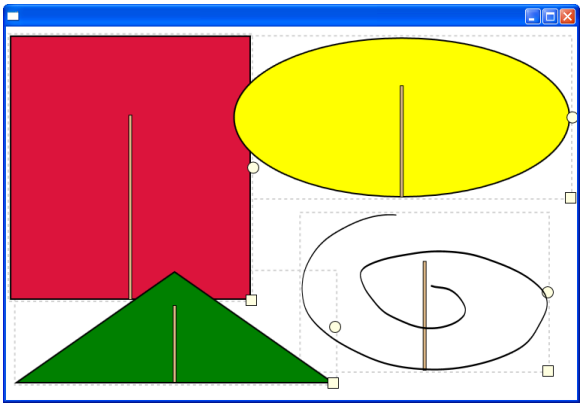

Figure 1: Handles appear when the stylus pauses in hover space.

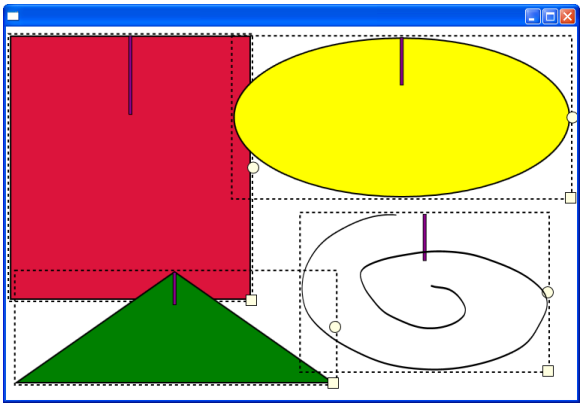

Figure 2: How selected objects appear.

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 the user from accidentally selecting intervening objects.

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. The user can also use the rotating and resizing handles, which we provide as a convenience for common editing tasks. Clicking and dragging these small shapes causes the shape to transform: the square will resize only the corresponding object, and the circle will rotate only the corresponding object. Unlike object movement, objects do not need to be selected to be rotated or resized, but rotating or resizing an object selects it. We based both of these interaction decisions on early user feedback.

The rotate and resize handles are not unique to HoverCross;

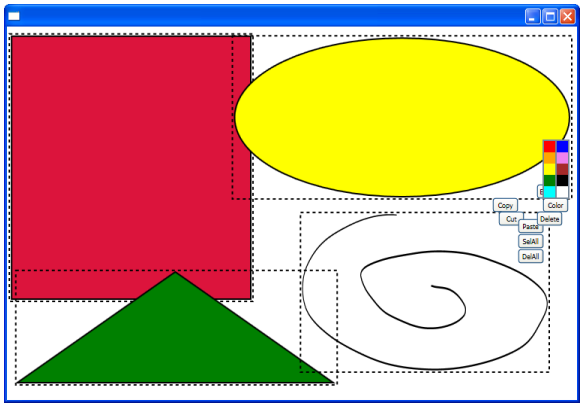

Figure 3: The context editing menu.

they mimic existing transformation techniques used by programs such as PowerPoint. However, their presence demonstrates that common transformation tools do not conflict with the HoverCross paradigm. Because selection occurs in the hover space, while transformation occurs when the pen is down, attempting to select objects will not accidentally transform them. Because the handles are small, users are unlikely to accidentally hit them when trying to draw content, even when the handles are invoked by accident.

For more complicated editing tasks (e.g. copy, paste, etc), HoverCross can be integrated with any type of menu. In our user studies we allowed editing through a context menu around the tip of the stylus (Figure 3), invoked either through a semicircle gesture or a "right-click" gesture (i.e. pressing and holding the stylus tip to the screen). The context menu has the advantage that with little practice the user can easily coordinate drawing, selecting, and editing in one fluid motion. However, designing an optimal menu style is tangential to our current research; HoverCross could easily be integrated with well-developed marking menus found in other applications [4, 11].

DESIGN DECISIONS AND FEEDBACK

The design of HoverCross focuses on giving users the ability to transition seamlessly between drawing and editing. Throughout our design we conducted several preliminary user tests to determine the effectiveness of HoverCross, and adjusted our design based on their results. In this section we discuss the interface's strengths, as well as how the weaknesses we discovered through our user tests led to the final design we present above.

Our user tests consisted of two phases. In the first phase, we tested our initial design with seven users. All users in this study were college students with some experience using a tablet computer, but no experience with hover or crossing-based interfaces. In this test, we compared three selection interfaces: our initial HoverCross design; a variation of HoverCross called Cross; and the traditional modal lasso select. Cross differs from HoverCross in that users cross objects with the pen on the screen while holding down a non-preferred hand button. 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. These tasks included simple selection and editing actions, using simple, non-overlapping shapes. We observed users and asked for their feedback.

Based on the results of our initial test we made substantial modifications to our interface, described below. The second phase of our user testing focused on guiding the continued modifications, and on evaluating the HoverCross's strengths and weaknesses on a more realistic, creative task. This phase involved seven new users (again, college students with the same experience as in the first phase), but instead of creating and editing specific diagrams, users were told simply to draw a picture. These pictures often became fairly complex and included shapes overlapping other shapes. In addition, rather than testing a static interface, we refined the interface between users when we sensed we were receiving consistent feedback from users so far.

Strengths

Our user tests consistently reflected that most users enjoyed the transition between sketching and selecting, without the hassle of pressing a button or other explicit indication, despite the reliability of the button. While some users reported that the interface was unintuitive and hard to use at first, they also said that the interface got easier after a few minutes of interaction. Users also described HoverCross as "interesting," "promising," and "fun."

Confirming previous results [1], we noticed that the crossing selection metaphor has several advantages over traditional lasso selection. For selecting single shapes or shapes roughly in a horizontal row, we observed users performing selection quickly and reliably. Some users commented that for small selections drawing a circle around the objects seemed excessive, and they preferred the crossing metaphor. Additionally, when selecting shapes spread out in space, users generally had no trouble learning to enter and exit hover space to avoiding selecting intervening objects. Users reported that they felt that selecting a few spatially separated objects with lasso select was awkward, while with HoverCross they could simply move the pen across the screen to the objects of interest.

A final strength of this interface is that any part of it may be implemented in conjunction with most other pen-based interface techniques. 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.

Design Modifications in Response to Common Problems

Handle placement. A common problem we discovered in the first user tests was that if handles are the same height as their shapes, they would either be too easily selected or too difficult to select if the shapes were either too tall or too short. So, rather than having the selection handles fit the shapes precisely, we gave the handles minimum and maximum sizes. We also made the handles switches that change position and color depending on whether their shape is selected to facilitate user interaction, so the handles could never extend across

the entire shape. Users in latter tests 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.

Occluded Shapes Another common problem we found in early user tests was the fact that shapes obstructed by other shapes would be either difficult or impossible to select. We fixed this problem by counting crosses over obstructed handles as legitimate crosses. This improvement consistently tested well when given to users for feedback. Although we worried that allowing crossing selection of obstructed shapes might produce accidental selections, users rarely cited it as a problem, in contrast with the negative feedback we received about the inability to select obstructed shapes.

Still, this improvement gave rise to another complication in the user tests, when shapes were grouped directly on top of other shapes. For example, if one handle became grouped directly on top of another handle, the user would switch them both simultaneously. 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.

Resizing and Rotating Our initial prototype did not allow users to resize or rotate objects. We quickly realized that these abilities were important to enable more realistic interaction with the system. We added the rotation and resize handles as described above, and users' responses were overwhelmingly positive. The only change we made to these handles based on user feedback was to allow users to rotate and resize an object without having to select the object first.

Future Improvements

Despite HoverCross's promise, there are several improvements we would like to explore. First, users occasionally complained about the selection boxes becoming too cluttered, particularly for diagrams with many small strokes; we could address this problem by grouping strokes or by changing the selection box paradigm for strokes, for example having the box fit the stroke more tightly. Second, most users found HoverCross cumbersome for selecting a large number of tightly grouped objects, where a lasso or box-select seemed more appropriate. As mentioned above, we could address this problem by integrating HoverCross with traditional lasso select, or perhaps by introducing a box or lasso-select ability directly in the hover space. Third, a number of user's comments focused on difficulties with the shape recognizer or invoking the context menu. These aspects are not central to the HoverCross paradigm, but improving them would allow users to focus solely on the utility of HoverCross.

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.

HoverCross determines whether or not a motion crosses a handle by storing the previous stylus location and passing it, along with the current stylus location, to an intersection test. This test checks whether the line segment between the two points crosses the center of any handles. If it does, the method returns all the handles that the motion crosses.

CONCLUSION

HoverCross combines many simple and effective ideas from recent advances in pen-based interfaces to provide an integrated interface for inking and editing. Used in combination with other pen-based interaction techniques, we believe 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

Removed for blind review.

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. K. Hinckley, S. Zhao, R. Sarin, P. Baudisch, E. Cutrell, M. Shilman, and D. Tan. Inkseine: In situ search for active note taking. In *Proc. of CHI '07*, 2007.
5. R. S. Kattinakere, T. Grossman, and S. Subramanian. Modeling steering within above-the-surface interaction layers. In *Proc. of CHI*, 2007.
6. 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.
7. 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.
8. E. Saund and E. Lank. Stylus input and editing without prior selection of mode. In *Proc. of UIST*, 2003.
9. S. Subramanian, D. Aliakseyeu, and A. Lucero. Multi-layer interaction for digital tables. In *Proc. of UIST*, 2006.
10. L. Tesler. The smalltalk environment. *Byte*, 6:90–147, August 1981.
11. R. C. Zeleznik, A. Bragdon, C. Liu, and A. Forsberg. Lineogrammer: creating diagrams by drawing. In *Proc. of UIST '08*, pages 161–170, 2008.