

Hover Widgets: Using the Tracking State to Extend the Capabilities of Pen-Operated Devices

Tovi Grossman^{1,2} Ken Hinckley¹ Patrick Baudisch¹ Maneesh Agrawala^{1,3} Ravin Balakrishnan²
¹Microsoft Research ²University of Toronto ³UC Berkeley
Redmond, WA Toronto, ON Berkeley, CA
research.microsoft.com www.dgp.toronto.edu www.cs.berkeley.edu
{kenh, baudisch}@microsoft.com {tovi, ravin}@dgp.toronto.edu maneesh@cs.berkeley.edu

ABSTRACT

We present Hover Widgets, a new technique for increasing the capabilities of pen-based interfaces. Hover Widgets are implemented by using the pen movements above the display surface, in the tracking state. Short gestures while hovering, followed by a *pen down*, access the Hover Widgets, which can be used to activate localized interface widgets. By using the tracking state movements, Hover Widgets create a new command layer which is clearly distinct from the input layer of a pen interface. In a formal experiment Hover Widgets were found to be faster than a more traditional command activation technique, and also reduced errors due to divided attention.

Author Keywords

Hover Widgets, pen input, gestures, tablets.

ACM Classification Keywords

H5.2. Information interfaces and presentation (e.g., HCI): User Interfaces, Graphical user interfaces.

INTRODUCTION

Pen-based interfaces are effective tools for a variety of tasks, such as freeform note taking, and informal sketch design. However, these devices typically lack the keyboard keys, buttons, and scroll wheels that can provide shortcuts for common tasks on the desktop. As a result, the user must zigzag the pen back and forth between the work area and the system menus. This slows users down and diverts their visual attention from their actual task at hand.

Localized user interface elements attempt to solve this problem by bringing the interface to the locus of the user's attention, as indicated by the current pen location [5, 10, 12]. A significant challenge for localized interfaces is that the user must invoke them somehow, such that a pen stroke on the screen activates the interface rather than leaving behind ink. Even with the use of a well-crafted gesture

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHI 2006, April 22–28, 2006, Montréal, Québec, Canada.
Copyright 2006 ACM 1-59593-178-3/06/0004...\$5.00.

recognition engine, unrecognized gestures can be misinterpreted as ink, and strokes intended as ink can be falsely recognized as gestures, causing unexpected results.

One approach to address this problem is to require the user to press a physical button to explicitly distinguish between command modes and an ink input mode [12, 14]. A button can provide an efficient and effective solution [11], but in some situations it is just not practical. Many mobile devices or electronic whiteboards lack a suitable button, and even if a button is available, it may be awkward to use [18].

We seek new strategies and techniques for supporting localized user interface interactions in pen interfaces. Many pen devices (such as Wacom Tablets and Tablet PC's) support a *tracking state*. The tracking state senses the pen location while the pen is proximal to the interaction surface. However, the literature offers few examples of uses for the tracking state other than cursor feedback [5, 7].

We propose *Hover Widgets*, a novel interaction technique that extends the capabilities of pen-operated devices by using the tracking state to access localized user interface elements. A Hover Widget is invisible to the user during typical pen use, but appears when the user starts moving the pen along a path in the tracking state, and then activates when the user reaches the end of the path and clicks the widget with the pen. For example, the user might form a backwards 'L' shape to activate a marking menu (Figure 1).

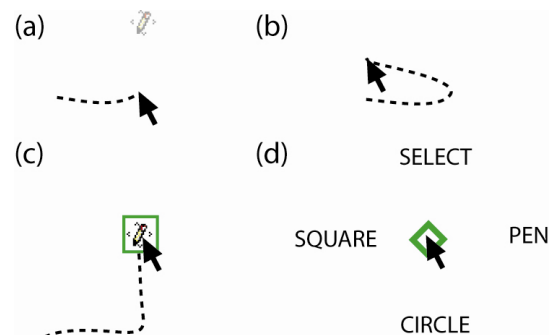


Figure 1: (a) When the user starts a Hover Widget gesture (here a backwards 'L'), the widget fades in. (b) The user exits the gesture, so the widget fades out (c) Upon completing the gesture, the cursor is over the associated Hover Widget. (d) The user clicks the widget to activate it. The dashed line is for illustration only, showing the pen's path in the tracking state.

Our research contributes a new way of using the pen tracking state to extend the capabilities of pen interfaces. We discuss the design space of Hover Widgets, consider various tracking state gestures which could be used, and explore various means for activating the Hover Widgets once the associated gestures are performed. In our prototype application, four 'L' shaped Hover Widgets are used to activate localized interactions, compensating for the absence of time-saving desktop items not available in pen interfaces, such as hotkeys, mouse buttons, and scroll wheels. In our studies, we found that an 'L' shaped gesture could be performed quickly, and had little chance of false activation. In a task sensitive to the user's focus of attention, we found that the localized properties of Hover Widgets made using them faster and also provided more accurate results compared to using a standard toolbar icon.

RELATED WORK

Hover Widgets combine three fields of related work: gestures, localized UIs, and uses for the tracking state.

A large amount of work exists in gesture-based systems for pen input [12, 14, 15, 18, 19]. These systems differ from the gestures used for Hover Widgets, as the gestures are carried out on the surface of the display. A documented difficulty associated with this technique is that the gestures can be confused with the data input, generally ink, causing unexpected results which must be undone [11, 15]. Even the most obscure gesture could be falsely recognized - if the user was illustrating the system's gestures, for example, then those illustrations would be recognized as the gestures which they illustrate. To alleviate this problem, some systems require users to explicitly switch between ink and gesture modes [12, 14]. In a recent study, it was shown that a button used by the non-dominant hand was most effective for this mode switch [11].

Other localized interaction techniques, such as pop-up menus, are generally activated with physical buttons. Two implementations of localized scrolling techniques which were recently developed supported scrolling as the only input mode, so their invocation was not an issue [13, 16].

The tracking state of the pen, when it is above the display surface, is one of the three states sensed by pen-based systems. The mouse can also be in the tracking state when it is moved without a button pushed [4]. Usually, this state is used to track the current position of the cursor, but there has been previous work using it for other functionality.

The Microsoft Windows operating system provides tool tips when users hover above an icon. These pop-up boxes display information about the icon, but cannot be clicked. A more interesting example is seen in the Windows XP Tablet PC Edition, which supports a gesture made in the tracking state. If the user scribbles above the display surface, a character entry tool pops up. Some users may find this feature irritating. It can be activated accidentally, and there is no visual guidance showing the user what to do for the gesture to be recognized.

In [8], users could share documents between multiple Tablet PCs by performing a drag gesture from one device to another called a "stitching" gesture. In one of the designs, this gesture could be done in the display tracking zones.

The tracking menu is an interactive interface widget which relies on tracking state actions [5]. The menu is a cluster of graphical widgets surrounded by a border within which the cursor moves. While the cursor is within the border, menu items can be selected via the usual cursor movements. However, if the cursor reaches the menu's border while moving in the tracking state, the menu is dragged around with the cursor. This allows for a smooth modeless transition between menu repositioning and menu item selection. Further, the contents of the menu are always in close proximity to the cursor. However, unlike Hover Widgets which are modeless, a tracking menu is modal in that an explicit action is required to dismiss it. As with all modal interface elements, while the tracking menu is active, only commands within that menu can be executed. This limitation of tracking menus is one motivation for exploring the design space of modeless interface widgets - like Hover Widgets - that leverage the tracking state.

HOVER WIDGETS

Design Properties

Hover Widgets offer a number of beneficial properties:

- **New Command Layer:** Hover Widgets use the tracking state to create a new command layer which is clearly distinguishable from the input layer of a user interface. A user does not need to worry about the system confusing ink and gestures.
- **Localized UI:** Hover Widgets are always local to the cursor, which may save the user time and reduce physical movements. For example, if a user needs to undo a mistake, instead of traveling to the top of the interface to click an icon, the user could make a gesture while hovering, and then click to activate an "undo" Hover Widget.
- **Undivided Attention:** Hover Widgets allow users to maintain their focus of attention on their current work area. If a user is reading the bottom of a page that they are annotating, a gesture in the tracking state could be used to activate a virtual scroll ring [13], allowing them to scroll as they continue to read. The user would not have to shift their attention to an icon on the border to initiate scrolling.
- **Button-Free UI Activation:** Hover Widgets provide a mechanism to quickly bring up other localized user interface elements, without the use of a physical button. For example, a Hover Widget could be used to activate a marking menu or virtual scroll ring.
- **Integration with Pen UI:** Hover Widgets can be integrated into pen-based user interfaces, allowing fast transitions between ink and commands. If a user noticed a mistake in a document while they were scrolling, they could lift the pen and then draw a circle around the mistake. The user could then activate the scroll tool to continue scrolling.

Design Space of Hover Widgets

Using the tracking state pen actions is a relatively new topic. As a result there are a number of issues we must consider in the design of Hover Widgets. We now discuss the design space of Hover Widgets that we explored, and the reasoning behind the design decisions which we made.

Shape

If a gesture in the tracking state will be required to activate the Hover Widgets, then the gesture must be easy to perform. At the same time, the gesture must not occur in natural tracking state movements. Otherwise, Hover Widgets would be activated unintentionally. This presents a trade-off between simplicity and ambiguity. If the gesture is complex, executing it will be slow. But reducing the complexity may increase ambiguity, causing unintentional activations.

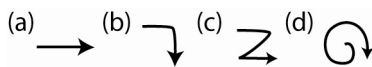


Figure 2: (a) Single level stroke. (b) Two level stroke. (c) Three level stroke. (d) Spiral stroke.

UniStroke characters [6] provide a good place to start when searching for appropriate gestures. The simplest gestures consist of a single directional stroke, but there are also compound stroke gestures with one and two corners (Figure 2a, b, c). Although the single-level strokes are simple, they will cause many false activations, as the pen only needs to move in a single direction to activate the widget.

The two-level strokes are less likely to cause false activations, so in our studies we focus on ‘L’ shape strokes, which have 90 degree angles. These strokes have minimal complexity, and we would not expect the sharp corners to appear in tracking state pen actions. We verified this intuition by simulating the Hover Widgets on captured pen data from internal users. We analyze the results of a more formal simulation study in Experiment 1.

While the two-level strokes may be the best shape strictly in terms of the simplicity-ambiguity tradeoff, there is no reason more complex strokes couldn’t be used. Along with three stroke compound gestures, we have also explored other stroke shapes such as spirals (Figure 2d). Although the strokes are more complex, they could be used to increase the vocabulary of an interface.

Gesture Recognition and Visualization

Two interrelated design issues for Hover Widgets are how they are visualized, and how the system recognizes them. The issues are associated because the visualization should convey to the user the exact requirement for either invoking the command or preventing the command from occurring.

Our strategy is to use gestures which are constrained and guided by boundary walls surrounding the target stroke, creating a tunnel that the user must traverse (Figure 3). The visual appearance of the tunnel defines the movements required to acquire the associated Hover Widget. The main

benefit of using such a simplified gesture recognition strategy is that users can quickly understand exactly what is required to activate a Hover Widget. Using the tunnel boundaries also makes the gesture recognition algorithm relatively simple. A more elegant recognition system could possibly improve performance, but it would be challenging to visualize complex gesture constraints.

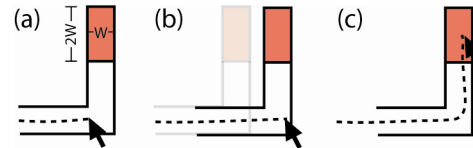


Figure 3: (a) A cursor moves through the Hover Widget tunnel. (b) The tunnel is repositioned if the cursor leaves its boundaries. (c) The Hover Widget can be activated once the cursor reaches the activation zone, shown in red.

If the cursor leaves the boundaries of the tunnel, then the origin of the tunnel is repositioned to the earliest point of the current hover stroke which could begin a successful gesture (Figure 3b). As long as a user’s stroke *ends* with the required movements, the Hover Widget will be activated. This makes, the ‘L’ shaped gesture semi-scale independent, as the first segment of the stroke does not have a maximum length (Figure 3c). A consequence of this algorithm, is that the sections of the tunnel boundaries act similarly to the borders of tracking menus [5]. The Hover Widget, however, is not simply an ‘L’ shape tracking menu, since intersecting with other sections of the tunnel will reset the origin of the tunnel, ensuring that *only* ‘L’ shaped pen movements can activate the Hover Widget.

Activation

Once the cursor travels through a tunnel, the associated Hover Widget can be activated. We have explored three methods for activation: *pen down*, *tapping*, and *crossing*.

With *pen down* activation, the user simply brings then pen in contact with the activation zone after completing a gesture in the tracking state. In initial user tests, we found that errors caused by overshooting the activation zone could be adequately prevented by making the region twice as long in the direction of movement (Figure 3a). The tunnel is reset if the cursor leaves this activation zone. *Pen down* is the default activation method in our application, and is the technique used in our experiments.

Tapping to activate the Hover Widgets is another option which we have explored. Instead of just bringing the pen in contact with the display, the pen quickly taps the display (i.e. a *pen down* event followed by a *pen up* event). This technique could be used to reduce false activations.

In the case of *crossing* activation, the Hover Widget is activated as soon as the pen crosses the end of a tunnel, while still in the tracking state. Implementing this technique increased the frequency of unintentional activations of ‘L’ shaped tunnels, but with more complex tunnels, such as spirals, false activations do not occur.

Visualization Techniques

Earlier we argued that recognition should be correlated to the way that Hover Widgets are visualized. While we observed that drawing the tunnels is beneficial when learning to use the Hover Widgets, seeing the tunnels at all times would become visually distracting, especially when the Hover Widgets were not in use. Expert users may not need to see the tunnel at all. In this section we outline strategies for visualizing the Hover Widgets such that the user sees what they need to see, when they need to see it.

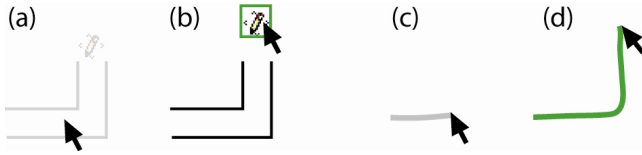


Figure 4: (a) The tunnel and activation zone fades in after 40% progress has been made. (b) The activation zone is displayed as a square icon. (c) The cursor trail visualization shows the path of the pen in the tracking state. (d) The cursor trail turns green when the cursor reaches the activation zone.

Both the tunnel and activation zone can either be displayed or hidden. When displayed, a fade-in point can be set, which defines how much progress must be made before the widget becomes visible. For example, a user may only want to see the activation zone or tunnel after they have progressed through 40% of the tunnel (Figure 4a). Once the cursor reaches the fade-in point, the widget slowly fades in. The activation zone is displayed as a square icon, which illustrates its associated functionality. Because the activation zone is actually rectangular, the icon will drag along with the cursor until it exits the region (Figure 4b).

We have also explored a dwelling fade-in, where the Hover Widget becomes visible if the pen dwells in any fixed location of the tracking zone. This is especially useful when multiple tunnels are present, so users can see which tunnel to follow to access a certain Hover Widget (Figure 5).

A final visualization technique which we have explored is the cursor trail. The path that the cursor has taken is shown, beginning at the tunnel origin, and ending at the current cursor location (Figure 4c). If the cursor completes the gesture, the trail turns green, indicating that the Hover Widget can be activated (Figure 4d).

Multiple Hover Widgets

Multiple Hover Widgets can be used simultaneously, with each one having its own parameters (Figure 5). When multiple Hover Widgets do exist, each individual Hover Widget is updated independently of the others. This approach ensures that each Hover Widget will still be activated if and only if its corresponding gesture is made. The relative positions of the 4 tunnels illustrated in Figure 5 are not locked. If the pen first moved to the right, it could still move up and then to the left to activate undo icon, without the need to first backtrack. This is a difference from Tracking Menus, which have a constant interface layout [5].

APPLICATION

We implemented a painting program, with all functionality of the application accessed via Hover Widgets. Hover Widgets are not limited to drawing applications; this context was used as a proof-of-concept prototype.

The system allows us to explore how Hover Widgets can replace desktop user interface elements using localized interactions. The only standard GUI element is a control panel that can be used to modify various parameters of the Hover Widgets. By using the Hover Widgets for *all* functionality we are pushing the limits of Hover Widgets, allowing us to see how far we could extend their use. In an actual application, the Hover Widgets would likely complement standard menus and tool bars. Four ‘L’ shaped Hover Widgets are used in the application (Figure 5). We now describe the functionality of these four Hover Widgets.

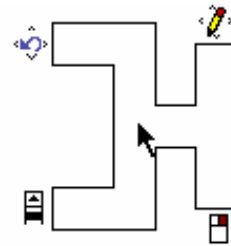


Figure 5: The four ‘L’ shaped Hover Widgets used in our application. Users would only see this entire “road map” if a dwelling fade-in occurred.

Tools Hover Widget

The tools Hover Widget (right-up) can be thought of as replacing an icon toolbar, found in most drawing applications. Activating the Hover Widget brings up a single level marking menu. From this menu, the following command selections are available: selection tool, pen tool, square tool, circle tool, and pen properties. The pen properties option pops up a localized CrossY Menu [2], allowing users to select the color and width of their pen.

Edit Hover Widget

The edit Hover Widget (up-left) replaces the standard “Edit” menu, by popping up a marking menu. Its options include the commands typically found in an application’s “Edit” menu: undo, redo, clear, cut, copy and paste.

Scroll Hover Widget

The scroll Hover Widget (down-left) allows users to scroll, without the need to travel to display borders. It can be thought of replacing the scroll wheel of a mouse. Activating this Hover Widget brings up a virtual scroll ring [13]. With this tool, users can make a circling gesture clockwise to scroll down, and counter-clockwise to scroll up.

Right Click Hover Widget

The right click Hover Widget (right-down) activates a right click tool. Once activated, the cursor is drawn as a right button icon. Subsequent pen down events simulate the functionality generally associated with clicking the right

mouse button. For example, clicking on a pen stroke pops up a marking menu, which provides options specific to that stroke, such as cut, copy, and properties.

Application Control Panel

The application contains a control panel which is opened by selecting a tab at the bottom right corner of the interface. This control panel allows users to explore the various Hover Widget settings and parameters. For example, users can switch to crossing or tapping activation, manipulate the width and length of the tunnels, and enable or disable the various visualization techniques. Parameters for the visualization techniques can also be modified, such as the fade-in point and the dwelling fade-in time threshold.

Informal User Feedback

We had roughly 10 computer literate users explore the application in short and informal demonstration sessions. While most users were able to use the Hover Widgets immediately, some needed a few minutes of practice. One of the most useful features seemed to be the dwelling fade-in. Users understood the required gestures when the tunnels were displayed, and could discover the various functionality of the application. All users liked the idea of using the tracking state for added pen input functionality.

EXPERIMENTS

Goals

The application which we developed allowed us to explore various uses of Hover Widgets in an interactive pen-based application. We now present a pilot study and two experiments designed to answer three main questions:

- (1) How do the parameters of the Hover Widgets affect their usability? The parameters we are primarily concerned with are the shape, orientation, tunnel width, tunnel length, and visualization technique (Figures 2, 4, 6).
- (2) How can these parameters be optimized to minimize false activations?
- (3) How do Hover Widgets with such optimal parameters perform in comparison to the status quo command selection techniques?

The first question, in itself, would require a great deal of experimentation, due to the number of parameters which may affect the Hover Widgets' performance.

To narrow down the number of variables in our experiments, we focused on a single value for some of these parameters. First, we only tested the 'L' shaped hover gestures, as they seemed to perform well in our initial observations. We set the length of the tunnel for these gestures to a value of 78 pixels, 39 pixels per direction. This length was found to cause few false activations in our preliminary captured data, and allowed users to complete the tunnels while resting their hand and moving mostly their fingers, with minimal movement from the wrist.

Pilot Study

We conducted a pilot study with six internal participants to determine appropriate values for the tunnel width and orientation for the full study. We expected the width (W), and length, or amplitude (A), to affect movement time (MT) as governed by the steering law [1]:

$$MT = a + b \left(\frac{A}{W} \right) \quad (1)$$

where a and b are empirically determined constants. The study also allowed us to get a quick sense of whether or not gesture orientation would significantly affect performance.

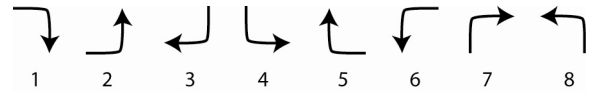


Figure 6: The 8 possible 'L' shape orientations.

Apparatus

The experiment was conducted on a Toshiba Portege 3500 Tablet PC running Windows XP SP1 Tablet Edition, with a 24.5 x 18.5 cm (1024 x 768 pixel) display. A stylus was used for input.

Procedure

A circle was placed in the center of the screen, which the user tapped to begin a trial. The user could then begin the required Hover Widget gesture, and the trial ended when the user successfully clicked on the Hover Widget activation zone. The tunnel and activation zone were displayed at all times, before and after a trial began. The cursor trail was also displayed once a trial began. The activation zone was displayed as a red rectangle, and turned green when the pen was above it.

Subjects were told to complete the trials as quickly as possible while minimizing errors. An error occurred when the pen clicked outside of the Hover Widget activation zone, in which case the user would have to start over from the beginning of the tunnel. Subjects were also told to avoid colliding with the walls of the tunnel.

Design

The tunnel length had a constant value of 78, and we tested 4 values of *Width* (12, 16, 20, and 24 pixels). The length of the activation zone (measured along the direction of movement) was set to two times the value of the *Width*. The *Width* values were fully crossed with all 8 possible *Orientation* values (Figure 6), resulting in 32 conditions. A repeated measures design was used - subjects completed 4 blocks, consisting of all 32 *Width-Orientation* combinations appearing 2 times each in random order.

Pilot Results and Discussion

In our analysis of movement time we discarded trials in which the tunnel was reset to the cursor position, due to a clicking error or a collision. A repeated measures analysis of variance showed a significant main effect for both *Width* ($F_{3,15} = 88$, $p < .0001$) and *Orientation* ($F_{7,28} = 3$, $p < .005$)

on movement time. Regression analyses showed that the data for all conditions fit the Steering Law (Equation 1) with an r^2 value of 0.98 (Figure 7). Although *Orientation* had a significant effect on movement time, the values only ranged from a minimum of 0.69 seconds for *Orientation* 5 and a maximum of 0.74 seconds for *Orientation* 8.

Overall, 11.6% of the trials had collisions, and only 0.78% of the trials had clicking errors. This is a good result, since a collision only indicates suboptimal performance, while a clicking error in an actual application could cause unexpected results. The collision rate was so high because of the condition where *Width* = 12, which had a 26.6% collision rate. With that condition removed, the average collision rate decreased to 6.7%.

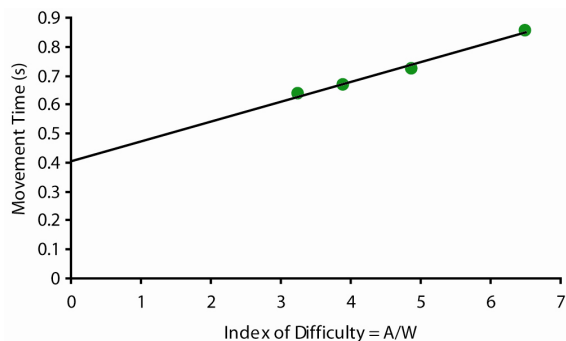


Figure 7: Movement time by the index of difficulty.

These results give us the required data to choose appropriate parameter values for our full study. We will include all 8 values of orientation to try to understand the cause for its differing effects on movement time. The pilot experiment, in addition to our preliminary captured data, shows that a width of 20 produces a good trade-off between minimizing movement time and error rate, while also preventing false activations. Therefore we used a width of 20 for the remaining studies.

Experiment 1

We now describe the first of two formal experiments which we conducted, testing how the orientation and visualization technique affect the Hover Widget performance.

Apparatus

The experiment was conducted on a Toshiba Portege M200 Tablet PC running Windows XP SP1 Tablet Edition, with a 24.5 x 18.5 cm (1024 x 768 pixel) display.

Participants

Twelve volunteers (two female, ten male) participated in the experiment. Participants were all right-handed, and controlled the stylus with their right hand. Five of the participants were experienced Tablet PC users.

Pen Capture/Warm up Session

Before starting the experiment, a twelve minute Tablet PC warm up session was given. This session was used for two purposes. First, it gave subjects who had never used pen

input systems a chance to get used to Tablet PCs. Second, it allowed us to capture pen data to verify our preliminary false activation findings. The capture sessions were broken up into three tasks, all using Windows Journal. Task one was to write a grocery list. In task two, participants drew directions to their house. In task three, they drew a picture of their dream house. At this point, participants had no knowledge of the Hover Widgets, and did not know their data would be used to test for false activations. Although only twelve participants went on to participate in the study, we captured pen data from fifteen participants, resulting in just over three hours of captured data.

Procedure

The procedure used for this experiment was similar to the pilot study. Above the starting circle, we displayed an arrow indicating the 'L' shape orientation for the current trial, both before and after the trial began. The tunnel still had a constant length of 78 and now also had a constant width of 20. The activation zone length was 40.

Design

We tested all 8 possible values of *Orientation*. We also tested the effects of the presence or absence of the *Cursor Trail*, and the *Fade-in Point*. The *Fade-in Point* values which we tested were 20% (early) and 70% (late), indicating the percentage of the tunnel that had to be traveled before the Hover Widget became visible.

A repeated measures within-participant design was used. A fully crossed design resulted in 32 combinations of *Orientation* (1-8), *Cursor Trail* (on, off), and *Fade-in Point* (early, late). Each participant performed the experiment in one session lasting approximately 20 minutes. The session was broken up by the 4 combinations of visualization techniques, with 5 blocks of trials for each of the visualizations. Each block consisted of all 8 orientations, appearing 2 times each in random order. Presentation order of the visualization techniques was counterbalanced using a Latin Square design, resulting in four orderings. Participants were randomly divided into 4 groups of 3, with each group performing one of the 4 orderings.

At the start of the experiment, participants were given a single warm-up block with the Hover Widget and cursor trail always visible, to familiarize them with the technique.

Results

We first discuss the results of the false activation analysis. We simulated the data from the pen capture sessions on all 8 possible 'L' shaped orientations. We report the results from two tunnel widths (16, 20 pixels), with all tunnels having a length of 78 pixels. Figure 8 shows the number of false activations which occurred in the 3 hours, broken up by orientation. Even at the larger width, orientations 6 and 7 had no false activations, while orientations 2 and 5 had 11 each. The captured data confirms our hypothesis that some gestures in the tracking state will rarely occur, but it also shows that corners in certain directions will be seen.

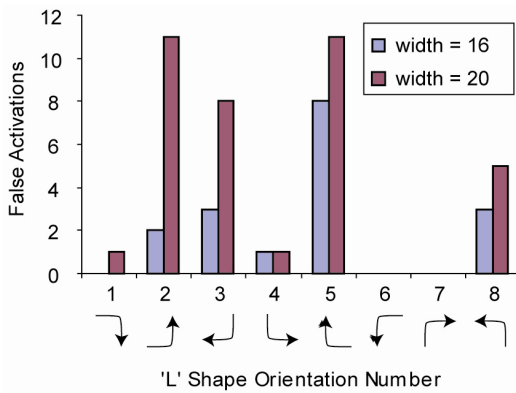


Figure 8: Number of false activations which occurred in the 3 hours of captured data.

We measured movement time (*MT*) defined as the time between lifting the pen after clicking to start the trial, and clicking the Hover Widget activation zone. We discarded trials in which clicking errors occurred in our analysis of *MT*. A repeated measures analysis of variance showed a significant main effect for *Fade-in Point* ($F_{1,11} = 38.24$, $p < .0001$), but not *Cursor Trail* or *Orientation*. With the early *Fade-in Point* the average movement time was 1.16s, and with the late *Fade-in Point*, the movement time was 1.30s.

We also found that the experienced tablet users performed significantly faster ($F_{1,11} = 292.42$, $p < .0001$). The average movement time was 1.41 seconds for non-tablet users, and 0.99 seconds for tablet users.

Clicking errors only occurred in 0.92% of all trials. This error rate was 1.4% for the late visualization, and 0.4% with the early visualization. This shows that even with the late visualization, the user had enough feedback to know when they could and could not activate the Hover Widget.

Learning Effects

Block number had a significant effect on movement time ($F_{3,33} = 6.15$, $p < 0.0005$), and a significant interaction with *Fade-in Point* ($F_{4,44} = 2.71$, $p < 0.05$) (Figure 9). Post hoc analysis revealed that the *Fade-in Point* only had a significant effect in block 1 ($p < .0001$). This result indicates that the early fade-in point would make for an effective training visualization, which the user could subsequently disable.

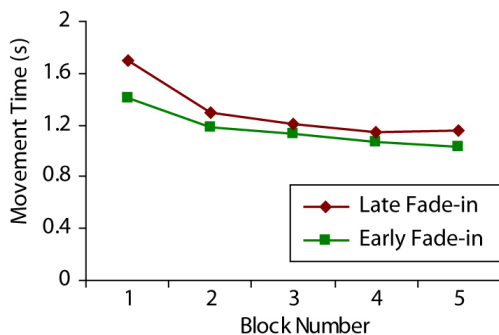


Figure 9: Movement time by Block and Fade-in Point.

Experiment 2

The results of Experiment 1 gave us a good understanding of how the parameters of the Hover Widgets would affect their performance in a controlled environment. Average movement times were comparable to those of 2-level simple and compound marking menus reported in the previous literature [10, 20]. The controlled environment, where the user performed all trials from a constant starting position, was well suited to answer our initial fundamental question about the Hover Widget parameters. Specifically, we found that the orientation did not have a significant effect on movement time, but did affect the number of observed false activations. Furthermore, we found that the cursor trail provided no advantage, while the early fade-in point significantly reduced movement time.

In Experiment 2, we explore the benefits gained from Hover Widget's property of being a localized interaction technique. We designed the experiment to allow us to analyze two potential benefits - faster access to commands and maintained focus on the primary task. The experimental task is an abstraction of real interface tasks which demand that the user focus their attention on a particular locale on screen, while at the same time requiring command selections. Examples of such tasks are scrolling or selecting a highlighter while proof-reading a document. Similar studies in two-handed input have been carried out exploring the effects of visual diversions during a drawing task [9].

Apparatus

We used a Wacom Cintiq 18SX interactive LCD graphics display tablet with a 32.9 x 29.9 cm (1280 x 1024 pixel) display. The display ran on a 1.4Ghz Windows XP desktop machine. This large-sized tablet allowed greater variation in the distance variable which we would be testing than the Tablet PC used in Experiment 1.

Participants

Ten volunteers (one female, nine male) participated in the experiment. One participant was left-handed, and all participants controlled the stylus with their dominant hand. Four of the participants were experienced tablet users.

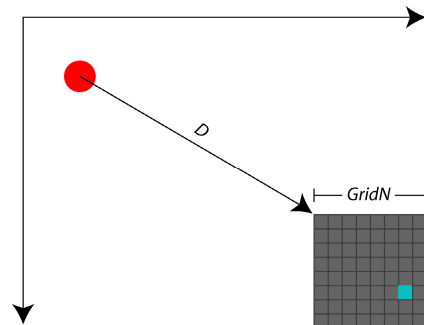


Figure 10: Experiment 2 setup ($GridN = 8$, $D = 300$). A trial was completed by selecting the blue target cell, which was only visible for 0.7 seconds. The red circle shows the position of the target needed to be selected in the icon condition. The dimensions of the grid ($GridN$) and its distance from the icon (D) varied. The top and left display borders are illustrated.

Procedure

At the start of a trial set, a square grid consisting of an array of 12x12 pixel square cells was displayed. The top left corner of the grid was aligned with the top-left to bottom-right diagonal of the display, at varying distances. The user clicked in the grid to start a trial, at which point a target cell in the grid would flash blue for 0.7 seconds. The target cell only flashed once, after which it returned to being visually indistinguishable from the other cells in the grid. To complete a trial, the user clicked this cell. Once the target cell was clicked, a new cell flashed, immediately beginning the next trial (Figure 10).

Before clicking the target cell, the user was required to make a successful command selection. Two command selection conditions were used. In the Icon condition, a single red circle was drawn in the top left corner of the screen, representing a traditional menu icon (Figure 10). The icon had a diameter of 24 pixels, which is about as large as any icon which appears in today's GUIs. In the Hover Widget condition, users were required to activate a Hover Widget before clicking the cell that had flashed. The Hover Widget parameters were set based on the results of Experiment 1. The Hover Widget tunnel had a width of 20, a length of 78, and 'L' shape orientation 7 (up-right). No cursor trail was displayed, and the fade-in point was 20%.

Since the target cell was only temporarily distinguishable from the surrounding cells, users had to attend to the grid when it flashed. Users would also benefit from attending to the grid *after* the target cell flashed, to reduce the reliance on their spatial memory of the target cell location, in order to successfully select it later. If participants were unable to recall the appropriate target cell and clicked a different cell, then they had to pause for 2 seconds, after which the target cell was highlighted to be clicked. This time penalty put it in the participants' best interest to select the correct target cell on their first attempt, allowing us to reliably correlate accuracy with maintained focus of attention.

Because gestures are generally not self-revealing [3] discoverability may be an issue with Hover Widgets. To explore this, participants were given no initial verbal instructions as to how to use the technique. Instead, a small text description and 40 second video clip were shown prior to using the Hover Widgets. This approach mimics what could be a start-up tutorial for first-time users of an application using Hover Widgets. If participants had subsequent questions, the experimenter would only repeat instructions that were already given in the video or text.

Design

A repeated measures within-participant design was used. Independent variables were command technique *CT* (*Icon*, *Hover Widget*), grid dimension *GridN* (4, 8, 12), measured as the grid side length in cells, and distance, *D* (300, 600, 900, 1200), measured as the distance between the top left corners of the grid and the display in pixels. A fully crossed design resulted in 24 combinations of *CT*, *GridN*, and *D*.

Each participant performed the experiment in one session lasting approximately 60 minutes. The session was broken up by the 2 command selection techniques, with 4 blocks appearing for each of the command techniques. Each block consisted of 24 trial sets, with each *GridN-D* combination appearing twice in random order. A trial set consisted of 5 trials on the same grid, with a different target cell for each of the 5 trials. Presentation order of the command techniques was counterbalanced with half the participants performing each technique first. A two minute warm-up session was given before each command technique.

Results

We removed subject 5 from the analysis, as the error rate for that subject was disproportionately high, and movement times were much faster, indicating that the subject was racing through the experiment. We also removed outliers more than 3 standard deviations from the group mean movement time. A total of 2.01% of the data was removed.

Movement Time

Movement time was defined as the time taken to select the target cell once a trial had started. Our analysis of movement time does not include trials in which an error was made during the command selection. Repeated measures analysis of variance shows significant effects for *CT* ($F_{1,8} = 1549, p < .0001$), *D* ($F_{3,51} = 167, p < .0001$), and *GridN* ($F_{2,34} = 499, p < .0001$), and significant *CT* x *GridN* and *CT* x *D* interactions ($p < .0001$). Overall movement times were 2.19s for *Icon*, and 1.76s for *Hover Widget*.

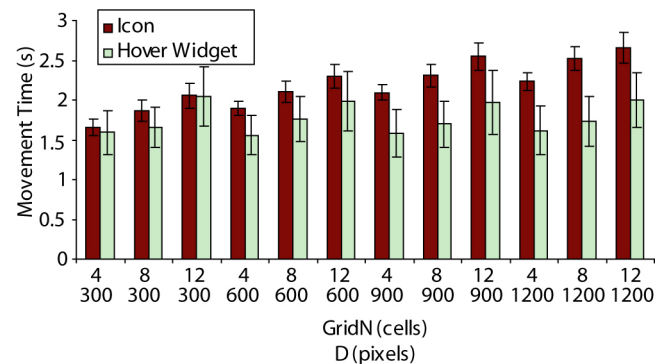


Figure 11: Movement time for the command techniques, by grid dimension and distance.

Figure 11 shows movement times for each of the 12 *GridN-D* combinations, illustrating the interactions. As expected, increased distance had little effect on the Hover Widget, while it increased times for the icon. Post-hoc analysis shows the Hover Widget technique to be significantly faster for every condition except at $D = 300$, where the differences were not significant for the 4x4 and 12x12 grids. This is a good result, showing Hover Widgets to be advantageous, even when the icon is near the grid.

It is interesting to note that *GridN* had an effect on the Hover Widget technique. This is in part due to the larger distances being traveled to get to the target cell. However the time to activate the Hover Widget once a trial started

was also significantly effected by *GridN* ($F_{2,16} = 123.28$, $p < .0001$). This was the result of users sometimes moving off the grid before activating the Hover Widget. We believe users did this to prevent their hand and Hover Widget from occluding the grid and target cell when it flashed.

As in Experiment 1, tablet experience significantly reduced Hover Widget movement times ($F_{1,9} = 31.1$, $p < .0001$), and also had a significant interaction with *Block* ($F_{3,27} = 6.92$, $p < .0001$). Post Hoc analysis showed that only in the first block were movement times significantly faster ($p < .0001$), demonstrating that, with practice, users new to Tablet PCs could use the technique just as well as experienced users.

Target Error Rate

We defined target errors as trials in which users selected the wrong target cell. Repeated measures analysis of variance shows significant effects for *CT* ($F_{1,8} = 166$, $p < .0001$), *D* ($F_{3,51} = 4.03$, $p < .01$), and *GridN* ($F_{2,34} = 166$, $p < .0001$), and a significant *CT* x *GridN* interaction ($F_{5,45} = 65.7$, $p < .0001$). Figure 12a illustrates the interaction between *CT* and *GridN*. The data shows that users have the ability to maintain their focus of attention on the grid while activating the Hover Widget, while selecting the icon causes divided attention. Surprisingly, with the icon technique, target error rate actually decreased with an increase of distance. This decrease may have been due to users spending more time focusing on the target cell before selecting the icon for larger distances. Regardless, users had trouble maintaining their focus of attention, even when the icon was close.

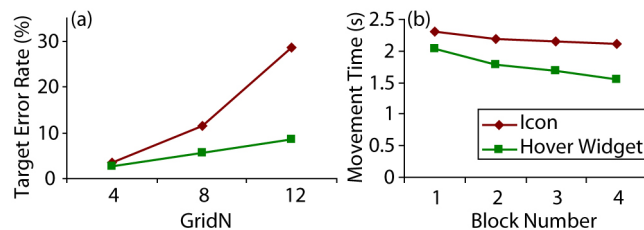


Figure 12: (a) Effect of grid dimension on the target error rate. (b) Movement time by block number, illustrating learning differences for the two techniques

Learning

We analyzed the effects of learning on each of the command techniques. Movement time was significantly affected by both *Block* ($F = 143$, $p < .0001$) and the *Block* x *CT* ($F = 25.3$, $p < .0001$) interaction. As can be seen in Figure 12b, learning is more apparent with the Hover Widget technique. Post hoc analysis shows that significant learning occurred after each block with the Hover Widgets, while the only significant change for the icon was between the first and second block ($p < .01$). Also of interest is that the Hover Widget was significantly faster in each of the four blocks, including the first ($p < .0001$).

Hover Widget Discoverability

To analyze the discoverability, we looked at the mean time to activate the Hover Widget in the warm-up trials, prior to which no verbal instructions had been given to participants

(see Procedure). Only one subject took more than 20 seconds for their first successful Hover Widget activation, and after the 6th trial the mean activation time across all subjects was under 3 seconds for each of the remaining trials in the warm-up. This data, in combination with the fact that Hover Widgets were significantly faster in the first block of the real experiment, shows that if proper system instructions are given, discoverability will not be an issue.

Trial Errors

We defined trial errors to be trials in which the user missed the icon or the Hover Widget. The error rate was 5.6% with the Hover Widgets, and 1.3% with the icon. Although the error rate for Hover Widgets is acceptably low, times may not have been as advantageous with equal error rates.

Hover Widget Collision Errors

We defined a collision error as any time the user aborted a Hover Widget after making at least 40% progress through the tunnel. Collision errors occurred in 4.1% of all trials. It is important to note that trials in which collision errors occurred were included in our analyses of movement time. The errors simply resulted in suboptimal trial performance.

DISCUSSION AND CONCLUSIONS

We have presented Hover Widgets, interactive widgets which are activated through simple gestures in the tracking state of pen-based interfaces. We have shown how Hover Widgets can be integrated into an application, replacing interface elements which can be inconvenient when using a pen. Users who tried the application liked the idea of using gestures while hovering, and were able to activate the Hover Widgets with little practice.

Furthermore, we have explored a method of pen interaction *without* the need for a button. Because our motivation for this is qualitative, as buttons can be unavailable or awkward to use, and not quantitative, our experiments did not compare Hover Widgets to button pressing techniques [11].

In an initial controlled experiment, we found acquisition times of Hover Widgets closely matched the movement times reported previously for similar simple and compound marking menu strokes. Our captured data showed that a number of these gestures will rarely occur unintentionally, some not seen once in 3 hours of pen usage. These results could drive the design of applications which use Hover Widgets. For example, we have modified our application to use orientations 1, 4, 6 and 7 to minimize false activations.

In the second experiment, we found that Hover Widgets reduced movement time and improved accuracy, when compared to a standard toolbar icon, in a task sensitive to the user's focus of attention. Movement time was significantly reduced in all conditions when the distance needed to travel to the icon was 600 pixels or greater, or about half the length of most Tablet PC displays. The analysis of accuracy showed that users were able to activate the Hover Widgets with a minimal shift of attention.

In Experiment 2, we also explored the discoverability of Hover Widgets by not providing any verbal instructions as to how to use them. Results showed that Hover Widgets could indeed be a walk-up-and-use interaction technique, if a start up tutorial is presented to first time users.

FUTURE WORK

The design space of Hover Widgets is very large, and there are a number of paths which we have not yet taken. More complex gestures could be investigated. The shape of the gestures could even represent their functionality. For example, the letter 'N' could be traced in the tracking state to activate a 'NEW' command Hover Widget. Sensiva Symbol Commander (www.sensiva.com) contains such gestures, but the pen must be down with a button pushed.

In the application that we developed, users manually controlled the parameters of the Hover Widget. This approach could be extended to allow users to design their own Hover Widgets, and customize their functionality.

Lastly, we believe Hover Widgets could be a beneficial technique for large display interfaces. In such setups, it becomes even more critical that a localized user interface is available, since the borders of the display may be difficult or impossible to reach. Generally, buttons are not available, but some electronic whiteboard technologies do sense the tracking state [17], so Hover Widgets could be used.

As for the controlled evaluation of Hover Widgets, Experiment 2 provided very positive results for the technique. However, the task was a simplified usage scenario, as only a single Hover Widget was used. We would like to explore what happens when the user must choose from multiple Hover Widgets, and how effectively Hover Widgets can be used to activate marking menus. We would expect to see similar benefits, with a possible overhead cost to learning multiple gestures.

It would also be interesting to evaluate the costs of performing gestures above the display surface. From our observations, the lack of force feedback when activating a Hover Widget was not problematic. However when combined with techniques which required the pen to be down, such as marking menus, some users needed practice before remembering that the pen had to be up for the Hover Widget, and down for the marking menu.

Overall, the Hover Widgets are a promising technique for extending the capabilities of pen-based interfaces. Our studies showed Hover Widgets to have strong qualitative and quantitative benefits which warrant future explorations.

ACKNOWLEDGEMENTS

We thank members of the ASI and VIBE groups at Microsoft Research and members of the DGP at the University of Toronto for insightful suggestions. We also thank David Thiel for producing the video, and all of our study participants.

REFERENCES

1. Accot, J. and S. Zhai. Beyond Fitts' Law: Models for trajectory-based HCI tasks. *ACM CHI 1997*, 295-302.
2. Apitz, G. and F. Guimbreti re. CrossY: a crossing-based drawing application. in *ACM UIST 2004*, 3-12.
3. Baudel, T. and M. Beaudouin-Lafon, Charade: remote control of objects using free-hand gestures. *Communications of the ACM*, 1993. 36(7): p. 28-35.
4. Buxton, W. Three-state model of graphical input. in *Interact.* 1990. Amsterdam: Elsevier Science, 449-456.
5. Fitzmaurice, G., et al. Tracking menus. in *ACM UIST 2003*, 71-79.
6. Goldberg, D. and C. Richardson, Touch-typing with a stylus. in *ACM CHI 1993*, 80-87.
7. Hinckley, K., et al. Interaction and modeling techniques for desktop two-handed input. in *UIST 1998*, 49-58.
8. Hinckley, K., et al. Stitching: Pen gestures that span multiple displays. in *AVI 2004*, 23-31.
9. Kabbash, P., et al. Two-handed input in a compound task. in *ACM CHI*. 1994, 417-423.
10. Kurtenbach, G. and W. Buxton. The limits of expert performance using hierarchical marking menus. in *ACM CHI 1993*, 482-487.
11. Li, Y., et al., Experimental analysis of mode switching techniques in pen-based user interfaces in *ACM CHI 2005*, 461-470.
12. Moran, T.P., et al. Pen-based interaction techniques for organizing material on an electronic whiteboard. in *ACM UIST 1997*, 45-54.
13. Moscovich, T. and J.F. Hughes, Navigating documents with the virtual scroll ring in *ACM UIST 2004*, 57-60.
14. Newman, M., et al., DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction*, 2003. 18(3): p. 259-324.
15. Saund, E. and E. Lank, Stylus input and editing without prior selection of mode in *ACM UIST 2003*, 213-216.
16. Smith, G.M. and m.c. schraefel, The radial scroll tool: scrolling support for stylus- or touch-based document navigation in *ACM UIST 2004*, 53-56.
17. SMART Tehcnologies, Digital Vision Touch Technology (2003): http://www.smarttech.com/dvit/DViT_white_paper.pdf
18. Zeleznik, R. and T. Miller, Fluid Inking: Modeless Integration of Gestures with Inking, in *Tech Report CS-05-10*. 2005, Brown University Computer Science.
19. Zeleznik, R., et al. SKETCH: An interface for sketching 3D scenes. in *SIGGRAPH 1996*, 163-170.
20. Zhao S. and R. Balakrishnan, Simple vs. compound mark hierarchical marking menus in *UIST 2004*, 33-42.