Handle Flags: Efficient and Flexible Selections for Inking Applications

Tovi Grossman
Autodesk Research

Patrick Baudisch
Microsoft Research

Ken Hinckley
Microsoft Research

ABSTRACT

There are a number of challenges associated with content selection in pen-based interfaces. Supplementary buttons to enter a selection mode may not be available, and selections may require a careful and error prone lasso stroke. In this paper we describe the design and evaluation of *Handle Flags,* a new localized technique used to select and perform commands on ink strokes in pen-operated interfaces. When the user positions the pen near an ink stroke, Handle Flags are displayed for the potential selections that the ink stroke could belong to (such as proximal strokes comprising a word or drawing). Tapping the handle allows the user to access the corresponding selection, without requiring a complex lasso stroke. Our studies show that Handle Flags offer significant benefits in comparison to traditional techniques, and are a promising technique for pen-based applications.

KEYWORDS: Pen input, selection, ink, Handle Flag, lasso.

INDEX TERMS: H.5.2 [User Interfaces]: Input, Interaction styles

1 INTRODUCTION

Pen-based systems allow for fluid and expressive input in tasks such as note taking and design sketching. However, the unique properties of pen-based interfaces, such as the lack of supplementary buttons, and the presence of many overlapping ink strokes, can make ink selection difficult [10]. *Handle Flags* offer a rapid and facile new technique for users to form ink selections, and thereby address two key problems with selection techniques.

The first problem arises because a system must decide whether a pen stroke should leave ink behind, or whether it should define a selection. A traditional solution is to employ a toolbar with an icon that the user taps to enter a lasso selection mode [18], but the resulting round trip time interrupts the users' attention from their work [7, 8, 12]. Furthermore, users must remember to return to inking mode, which slows performance and risks mode errors [12, 20, 28]. Even if a pen barrel button or tablet bezel button is available, mode errors occur if the user forgets to press the button, or trips the button while writing [11, 15]. Research has also explored implicit mode switching [25, 29], but such approaches can be prone to recognition errors and may not outperform the status quo techniques [5].

The second problem is the need to explicitly delineate the desired strokes. Lasso selection in a dense page of ink may entail a constrained, error prone steering task [1, 14]. For example, a user may lasso-select and move a word, only to leave behind a single stroke, such as the dot above an *i*. But if the system automatically parses and groups ink strokes (e.g., Microsoft Windows Journal), when the parser fails, it may prevent the user from selecting a word without also selecting undesired annotations that are nearby.

Handle Flags address these problems by judiciously fading in localized widgets while the user is inking (Figure 1). The user acts on a selection by tapping on an associated handle, which activates a radial pop-up menu. Handle Flags thus enable rapid selection of strokes, and activation of commands, without a hardware button or a round trip to a lasso mode icon.

In this paper, we outline the challenges associated with pen-based interfaces, and the previous research attempting to address these challenges. We then describe the design of Handle Flags, and discuss the key qualitative properties which they possess. This is followed by a description of the studies which we carried out to evaluate Handle Flags. The results of our studies indicate that Handle Flags can be a useful addition to pen-based user interfaces.

2 RELATED WORK

2.1 Selection in Pen-Based Interfaces

Ink selection can use tapping [17, 26], crossing [2, 23], or lassoing [4, 13]. It can be difficult for users to cross or tap every individual stroke in a word or drawing, but tapping works well for scattered discrete targets [17]. The speed of lassoing depends on the selection's complexity [1, 14]. For example, lassoing within a dense surround of notes is tedious, because steering [1] along a winding path is slow and error prone for lassos with accuracy constraints [14]. Handle Flags, by contrast, are much less dependent on the size of the selection and the density of surrounding strokes.

Some systems use automatic parsing to select high level structures in ink documents [3, 21]. Simple heuristic-based grouping algorithms, with appropriate feedback, provide useful functionality in Tivoli [18] and Flatland [19]; we implement a similar grouping heuristic for Handle Flags.

Unfortunately, when objects are not parsed correctly, automatic grouping can be annoying. For example, the system may erroneously group a stroke with a word when it actually belongs to a nearby diagram. If tap selection or lasso selection only offers the automatically formed group associated with a stroke (e.g. Windows Journal), then there is no way for the user to exclude the undesired stroke(s). Mankoff et al. present an impressive set of interaction techniques to resolve input ambiguity for recognition systems in general [16], which Handle Flags build upon.

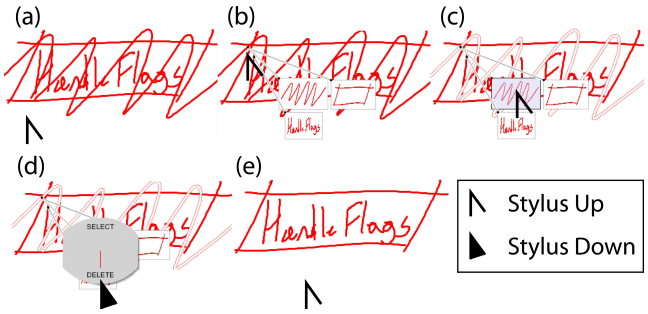

Figure 1. a) Multiple ink strokes. b) Handle Flags appear when the pen moves over ink. (c, d, e) Tapping on a handle selects and acts on the associated stroke(s) via pop-up menus.

tovi.grossman@autodesk.com
baudisch@microsoft.com
kenh@microsoft.com
Graphics Interface Conference 2009
25-27 May, Kelowna, British Columbia, Canada

PerSketch [27] stores an *Object Lattice*, where each element in the lattice represents an alternative parsing of the strokes. Any perceivable object corresponds to an element within the lattice. The system predicts which element in the lattice a user's selection gesture corresponds to. Follow-up work investigated the challenge of selecting the correct group among automatically generated selection candidates [24, 26]. Users could tap a stroke to cycle through the possible selections corresponding to that stroke. Saund et al. state that this method is adequate when the number of potential groups is small, and when the desired selection is available [26]. However, they also state that how to present multiple selection options to the user, as well as the use of graphical representations, both bear further investigation [24, 26]. Saund et al. implemented the multiple tap technique within a mouse-based image editor that did not support ink input, with right-click to access commands. The technique does not support transitions between inking and selection modes without requiring a round trip to command icons or the use of additional hardware buttons. Handle Flags adopt a similar strategy to the multiple tap technique, but provide a localized transition between inking and selection, and offer the user more explicit and graphical access to the underlying selection lattice.

The Office 2007 Mini Toolbar provides a localized contextual toolbar which fades in to allow the user to perform commands on a selection. However, this toolbar only fades in after a selection is made using the traditional methods. Another related technique is smart tags, used in some applications such as Microsoft OneNote 2003. A tag appears at a stroke as the pen approaches, and the user can tap it to access the associated selection. Unfortunately the design space of such techniques has not been investigated, resulting in a number of limitations which have yet to be addressed. For example, associated selections are determined by automatic parsers and cannot be modified; it can be difficult to know what a tag's associated selection is; if the smart tag is in a static location then it is impossible to begin an ink stroke at that location. Smart tags were removed from OneNote 2007, possibly due to such difficulties. In this paper, we explore the design space of localized selection widgets, resulting in our new design, Handle Flags, which differ significantly from smart tags and address their limitations.

2.2 Mode Switching in Pen-Based Interfaces

Many pen interfaces require users to switch between inking and selection modes, which can be prone to mode errors [20, 28]. A toolbar icon for lasso selection is a common approach, but this is time consuming and deflects the user's attention from his primary task [7, 8]. Li et al. compare several techniques for mode switching and conclude that a button controlled by the non-dominant hand works best [15].

Button-free alternatives have also been explored. Some gestural interfaces implicitly interpret the user's pen strokes [25, 29] to avoid explicit mode transitions. It is not clear if such interfaces can provide benefits without also introducing hidden states, delayed feedback, or specific orderings of operations that may be unclear to the user. For example, Deming and Lank found that an explicit mode switch is faster and preferred by most users in comparison to an inferred mode protocol [5].

The tracking menu [7] is a floating widget that gives users localized access to commands or modes, but only in a mode that the user must explicitly enable, and that the user cannot fluidly integrate with ink input. Hover Widgets use gestures in a pen's tracking state to perform global commands and mode switches [8]. Handle Flags contribute a new technique, focused on selection and contextual commands, which is faster than a round trip, and that can be as fast as or faster than even the non-preferred hand button technique.

3 Handle Flags

Handle Flags are a new technique that addresses the difficulties noted above. The main idea of Handle Flags is to provide a handle for accessing each *potential selection* in an inking application. When the pen approaches such selections, the Handle Flags fade in, offset from the pen location. The user can then tap on the handle that is associated with a desired selection, to select it or perform a command.

3.1 Design Principles

We make a number of careful considerations in the design of Handle Flags, resulting in key qualitative properties:

Button-Free: Researchers have argued that designing button-free interaction is important for pen input [8, 29]. In the realm of an inking application, Fitzmaurice et al. [6], and Guiard [9], have both made the point that writing itself is a two-handed task, with the non-dominant hand controlling the frame of reference of the paper. As a result the non-dominant hand may not be available to manipulate a control. Practically speaking, the non-dominant hand is often needed to hold the device in a mobile setting. Barrel buttons (on the stylus itself) typically require the user to shift their grip on the pen, interrupting the user's input flow, and can easily be hit accidentally, causing unexpected results. As such, Handle Flags were designed to be button-free.

Explicit and Integrated Functionality: Explicitly moded interfaces are prone to errors [20, 28], while gestural techniques that use an inferred mode protocol are prone to misinterpretation [5]. Tapping on a Handle Flag integrates the transition to selection mode and the definition of the selection scope into a single action. This can potentially reduce the risk of mode errors, while still allowing the user to explicitly communicate their intentions.

Localized: Previous research has demonstrated the drawbacks of requiring round trips to non-localized widgets [7, 8]. As such, Handle Flags were made to be localized, appearing in place when the stylus approaches ink.

Flexible Selection within Structured Scopes: In systems that use traditional groupings, it can be difficult to select individual elements that belong to a group, and users are often required to first use an "ungroup" command. Similar to PerSketch [27] and ScanScribe [24], Handle Flags are provided for multiple potential selections, allowing the user to select their intended scope.

Support Selection within Overlapping Ink: Selections in standard inking applications can be difficult due to overlapping and intersecting ink strokes. Handle Flags are positioned using a layout algorithm to guarantee that no handles occlude one another, even if the associated selections do.

Support Compound Selections: Without supplementary buttons, it can be difficult to define whether a selection should be added to a current selection scope, removed from the current selection scope, or be used to begin a new selection scope. Building upon GEdit [13] and Tivoli [18], Handle Flags support such compound selection operations.

3.2 Fade-In Heuristic

To prevent screen clutter, Handle Flags only appear when the system believes that the user wants to access them. The system determines this based on a simple algorithm using the spatial location and velocity of the stylus. If the stylus drops below a threshold velocity while in the tracking state, then handles for all ink strokes within 10 pixels will fade in. Using a velocity threshold rather than a dwell timeout threshold [15] reduces the need for users to wait for a desired handle to appear. We use a threshold velocity of 333 pixels/second, which, in informal usage observations, was found to be adequately responsive while at the same time limiting unwanted appearances.

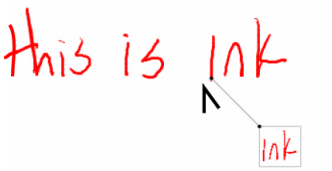

Figure 2. Handle Flags consist of a line, starting from the point on the ink closest to the pen, and a handle, at an offset location. The handle is rendered as a thumbnail image of its associated selection.

3.3 Placement

If an unwanted handle appears, it may occlude an area where the user wishes to begin an ink stroke. Even worse, if the handle appears just before the pen begins a new ink stroke, the user could tap it accidentally. To reduce the chances of this, we offset the handle from the pen position. A line originates from the point on the ink stroke closest to the pen position, but the actual handle, where the user would tap to activate it, is always displaced from its associated object– thus the name Handle Flag (Figure 2). Handles are displaced 50 pixels down and to the left for right-handed users, and 50 pixels down and to the right for left-handed users. This prevents occlusion from the user's hand.

3.4 Appearance and Activation

To aid the association between handles and ink strokes, we render the handles as thumbnail images of the strokes that they represent. The dimensions of the handle are proportional to the bounding box of the ink (Figure 2). The user activates a Handle Flag by tapping on this handle. Once this occurs, a radial pop-up menu is displayed. Within this menu, the user can select the associated ink strokes, or perform commands on them.

3.5 Fade-Out Heuristic

The fade-out heuristic must be sensitive enough that a user can dismiss an unwanted handle without interrupting the flow of interaction. However, the heuristic cannot be so sensitive that desired handles unexpectedly fade-out. The heuristic that we converged upon causes a Handle Flag to fade out if, over any period of time, the distance between the pen and handle increases by more than 35 pixels.

3.6 Multiple Objects

When the pen drops below the threshold velocity in an area containing multiple ink strokes, a separate Handle Flag for each of these objects will be displayed. The Handle Flags are placed using a layout algorithm that guarantees no two handles overlap. This gives the user the ability to easily select objects that would be difficult or impossible to select using traditional techniques.

For a left-handed user, the system first tries to position each Handle Flag at the default location – 50 pixels down and to the right. If this position causes an occlusion with a previously placed handle, the system traverses through a set of ordered candidate positions, until a position that does not cause an occlusion is found (Figure 3a). In essence the algorithm tries to choose the closest available position which is down from and/or to the right of the default location. While a handle is visible, its position is locked.

With all handles being down and to the right users can potentially move towards this area while searching for their goal handle. This is not possible in a technique such as splatter [22], where the objects encircle the pen location. In scenarios where the pen location is close to a border of the screen, the default offset vector can be modified to ensure that the Handle Flags remain within the window borders.

(a)

1	2	5	10
3	4	6	11
7	8	9	12
13	14	15	16

(b)

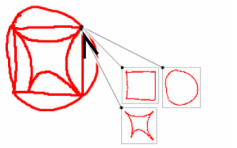

Figure 3. Layout for multiple objects. a) Traversal order, with position 1 being the default location. b) Handle Flag layout for 3 potential sections.

3.7 Working with Hierarchies

In addition to selection of individual strokes, Handle Flags can be used to select compound objects, such as words or diagrams. An individual stroke could be a member of multiple compound objects. That is, each individual stroke can be an element of multiple *potential selections.*

To give the user the flexibility to select and work with any of these potential selections, we provide a Handle Flag for each one. Unlike traditional *groups*, potential selections do not need to form a strict hierarchy. For example, a stroke *B* could belong to potential selections *AB* and *BC*. Three types of potential selections can exist:

1. Selections created and accessed by the user
2. Selections inferred by the system
3. Individual ink strokes

Selections of type (2) are the groups that some inking applications automatically form based on the structure of the user input [3, 21]. Our system, which groups strokes based on their spatial proximity, is similar to the algorithms used in previous systems [18, 19]. We chose this technique over more complex machine learning algorithms, since our interest lies in the interaction design, and because Handle Flags do not rely on accurate inferred groupings.

The Handle Flags for all potential selections are stored in a ranked priority list. The priority rankings come from the above numbering of the three selection types. Selections that were most recently used are given highest priority, followed by selections inferred by the system, with individual strokes given lowest priority. Each time the user employs a selection, it is brought to the front of the list.

When the pen hovers over an ink stroke that belongs to multiple selections, only the Handle Flag for that stroke with the highest priority is displayed, at the *root level*. Selections lower on the priority list are organized into a hierarchy which originates from the *root level.* Handle Flags which represent a subset of the individual strokes in the root level are stored lower in the hierarchy, and Handle Flags which represent a superset of the individual strokes in the root level are stored higher (Figure 4).

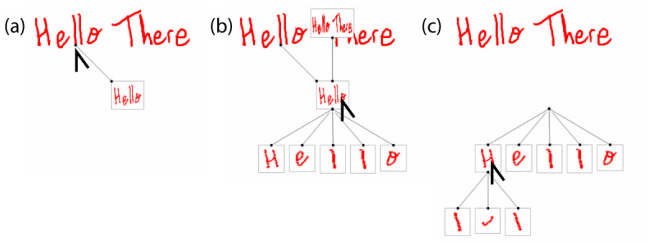

Figure 4. Handle Flag hierarchies. a) Only the Handle Flag with highest priority is displayed at the root level. b) Hovering over the root level Handle Flag reveals superset handles above the root, and subset handles below the root. c) The process can be repeated to traverse the entire hierarchy.

Initially, this hierarchy is invisible to the user, and only the root level fades in (Figure 4a). To access the remaining hierarchy, the user hovers over the handle at the root level. Using the same fade-in heuristic, this handle then expands (Figure 4b) to show the next level of the hierarchy, both above and below. The user can repeat this process to traverse the entire hierarchy (Figure 4c). To prevent users from getting lost in an endless path, only the root level Handle Flag expands both upwards and downwards. Remaining handles above the root only expand upwards, and remaining handles below the root expand downwards.

Because the system promotes recent selections employed by the user to the top of the priority list, it is fast for the user to repeat a selection, as the desired Handle Flag will always be at the root level. It is only when a user wishes to access a new selection that the cost of traversing the selection hierarchy is incurred.

3.8 Compound Selections

From a Handle Flag's menu, the user can choose three selection options: *New Selection*, *Add to Selection*, and *Remove from Selection*. This allows users to create new selection sets that do not already have a Handle Flag.

Alternatively, the user can also employ a traditional lassoing tool to do so. It is important to note that we do not intend Handle Flags to be a replacement to the lasso tool, but rather a tool to compliment lasso selection. Handle Flags are most effective when a desired selection already exists, but the lasso tool can be used otherwise. To provide localized access to the lasso tool, a small lasso icon in the top right corner of each Handle Flag fades in, if the user hovers over the handle for 0.4s. During the dwell period, users can first check if their desired Handle Flag exists (Figure 5a-c). If it does not, then they can tap on the lasso icon (Figure 5d) and make the desired lasso selection (Figure 5e). Thus, the user can habitually move to the Handle Flag to initiate all selections.

When the user makes a selection, the system outlines the selected strokes, draws a dashed bounding box around the selection, and attaches a selection handle to a corner of the bounding box border (Figure 5f). Tapping on the selection handle brings up a radial menu with options that act on the entire selection. The system also adds a Handle Flag for the newly defined selection, and brings it to the top of the priority list, so it will subsequently be available.

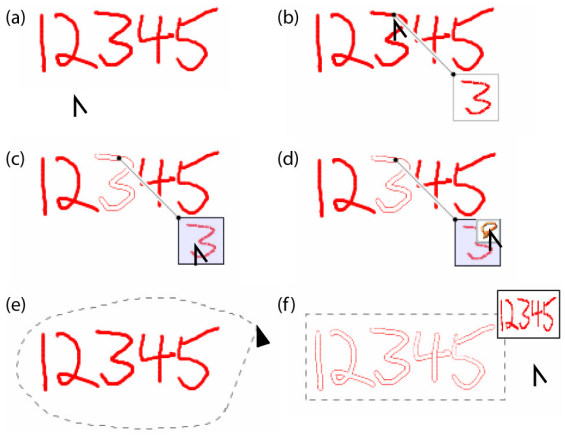

Figure 5. a) A user wishes to select "12345" b) The user hovers over "3" to display the root level Handle Flag. c) The user hovers over the Handle Flag, but no further Handle Flags are expanded. d) Instead, a lasso icon fades in. e) The user taps the lasso icon and then draws a lasso to make the desired selection. f) A bounding box is drawn around the selection, with a selection handle attached to its border.

4 Experiment 1: Abstracted Environment

In this experiment we investigate the potential quantitative benefits of Handle Flags, by obtaining an initial understanding of their underlying mechanics in comparison to existing techniques. To do so, we use an abstracted task in a controlled environment. While this will provide important data which could be relevant for future design considerations, the results cannot be used to draw firm conclusions about the technique in actual application use. We will investigate this practical issue in a second experiment which is performed in a more realistic usage setting.

4.1 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 3.6Ghz Windows XP desktop machine. A stylus with its barrel buttons disabled was used for input. For the non-dominant hand button technique, which is described in the Procedure section, a standard QWERTY keyboard was placed on the appropriate side of the tablet, based on the user's handedness.

4.2 Participants

Twelve volunteers (eight male, four female), aged 23-35, participated in the experiment. One participant was left-handed, and all participants controlled the stylus with their dominant hand.

4.3 Procedure

The task environment consisted of a green start circle, a 6x6 grid of X's representing individual strokes, and a red end circle. Only the 16 internal black X's were candidates for the selection. The bordering grey X's, were only used as distracters to control the spacing between the selections and surrounding targets (Figure 6a). Using X's allowed us to carefully control these spacings.

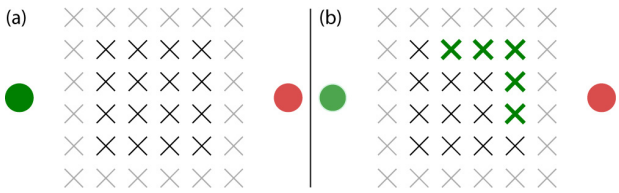

Figure 6. a) The experimental environment. b) Goal targets turn green after the user taps the start button.

To begin a trial, the user tapped on the start target, and a set of the X's from the grid would turn green. One of three possible levels of selection complexity (*Complexity*) was presented: either a single target, a row or column of 4 targets, or an 'L' shape of 5 targets (Figure 6b). We varied the spacing between targets (*Spacing*), at levels of 25, 50, or 75 pixels, measured by the distance between the target centers. The targets slightly overlapped in the 25 pixel spacing condition. To complete a trial, the user had to select the green targets using one of the four techniques described below, perform a command to revert their color to black, and tap on the red end target. We instructed users to perform the task as quickly as possible, while minimizing errors.

4.4 Lasso Icon Technique (*icon*)

For this technique the user had to first tap on a 48x48 pixel lasso icon in the top left corner of the display, about 570 pixels from the start location, to enter the lasso mode. This approximates the distance required to travel from the center of an average sized Tablet PC to the display border. The user could then lasso the goal target or targets with a single stroke. The criterion for selection was that 50% of each target in the selection had to be contained by the region defined by the lasso. After completing a successful

lasso, a 48x48 square handle would appear, centered directly under the stylus. This "handle delimiter" has been shown to be an efficient way to integrate selection with command activation [10, 13]. Tapping on this handle would activate a radial menu, with the revert command in the north direction. Users dragged the pen towards this option and then released to execute the command. The user could then tap on the end target to complete the trial.

4.5 Non-Dominant Hand Button Technique (*button*)

With this technique, the user entered the lasso mode by holding down the *Control* key on the keyboard. While holding the button down, the user could lasso the selection. Everything else was the same as the previous technique, except for the added constraint that the control button had to be released before tapping on the end target. While our goal was to design a button free technique, we included this technique in the experiment to see how well Handle Flags would compare to this previously studied technique that is known to be efficient [15].

4.6 Level 1 Handle Flag Technique (*handle1*)

In both Handle Flag techniques, Handle Flags were available for each of the 36 X's. Furthermore, the internal 4x4 grid of X's was grouped by row and by column, for a total of 8 more Handle Flags. In the 5 target L-shaped condition, a corresponding Handle Flag was available. This resulted in either 44 or 45 available Handle Flags in each trial. For the *handle1* technique, the priorities of the Handle Flags were initialized such that the Handle Flag for the goal target was always at the root level. Tapping on it activated the same pop-up menu used in the lasso techniques. At this point, everything else was the same as the previous techniques.

4.7 Level 2 Handle Flag Technique (*handle2*)

With this technique, the goal target appeared in the second level of the Handle Flag hierarchy. For example, if the goal selection was a row, the user would first have to hover over an 'X' in that row, and then hover over the Handle Flag for that individual 'X' to display the Handle Flag for the row. The user could then tap on the goal Handle Flag and continue as in the previous techniques.

4.8 Experimental Design

A repeated measures within-participant design was used. The independent variables were *Technique* (*icon*, *button*, *handle1*, *handle2*), *Complexity* (*single*, *line*, *l-shaped*), and *Spacing* (*25*, *50*, *75*). A fully crossed design resulted in 36 conditions. Participants completed the experiment in one session lasting approximately 50 minutes. The session was broken up by the four techniques, with two blocks of trials appearing for each of these techniques. Each block consisted of the 9 combinations of *Complexity* and *Spacing* in random order, repeated 4 times each, for a total of 36 trials per block. Presentation order of the techniques was counterbalanced using a 4x4 Latin square, with 3 participants randomly assigned to each of the four orderings. Before the first block of each technique, a 2 minute warm-up session was completed, to familiarize participants with the task.

4.9 Results

Trial completion time was defined as the time between releasing the pen after tapping the start button, and pressing down the pen on the end button. We discarded trials in which errors occurred (16.1% of trials), as there was no significant effect of *Technique* on the error rate.

Repeated measures analysis of variance showed main effects for *Technique* ($F_{3,33} = 58.5$, $p < .0001$), *Complexity* ($F_{2,22} = 235.7$, $p < .0001$), and *Spacing* ($F_{2,22} = 4.25$, $p < .05$). Average trial completion times were 2.21s for *handle1*, 3.03s for *button*, 3.25s for *handle2*, and 3.78s for *icon* (Figure 7a). Post hoc analysis using Bonferroni adjustment showed that *handle1* was significantly faster than all techniques ($p < .0001$), and that *button* was significantly faster than *icon* ($p < .0001$).

These results are important. First, they demonstrate, that the mechanics of Handle Flags can significantly reduce performance times. Further, the results show that there is about a 1s cost associated with traversing levels in the Handle Flag hierarchy. However, even with this additional cost, the *handle2* technique offered comparable results to the *button* technique (no significant difference, $p = 0.919$), and was a 0.5s faster than the button-free *icon* technique, although this did not reach significance ($p = .055$).

Both the *Technique X Complexity* ($F_{6,66} = 90.5$, $p < .0001$) and *Technique X Spacing* ($F_{6,66} = 3.10$, $p < .01$) interactions were significant (Figure 7b, c). Changing the complexity of the selection had little effect on the Handle Flag techniques, but significantly affected the lassoing techniques. A similar but lesser effect was observed with the *Spacing*. This demonstrates another important benefit regarding the mechanics of Handle Flags - the performance times are robust to the complexity of the selection.

5 Experiment 2: Sketching Environment

The results from the first experiment show that Handle Flags have the potential to be a beneficial technique for inking applications. However, this experiment was focused on understanding the mechanics of the technique, and thus, a number of simplifications were made for control purposes. First, the targets were abstract objects, and not real ink strokes from actual sketches. Second, for the Handle Flag techniques, users knew before each trial what level the goal Handle Flag would be at, potentially eliminating a cost of searching through the hierarchy. Finally, for the Handle Flag technique, the goal selection was always available from a Handle Flag, which eliminates a cost of users deciding whether or not they should even use a Handle Flag for the selection.

In this experiment, we investigate these issues in a more realistic usage setting. The overall task remains identical. However, selections are made from actual sketches. Furthermore, we integrate trials with the goal Handle Flag at the root level, at the 2nd level, and not available at all.

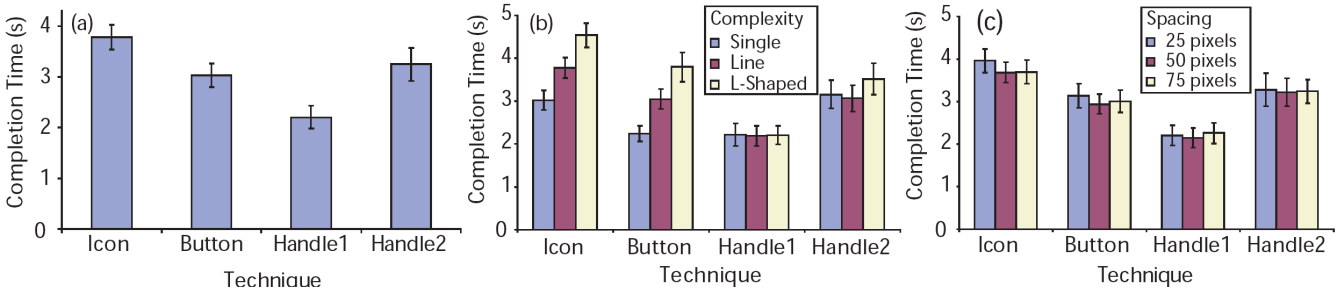

Figure 7. Completion times by (a) *Technique*, (b) *Technique* and *Complexity*, and (c) *Technique* and *Spacing*.

5.1 Apparatus

The experiment was performed on a 2.0 GHz Toshiba M400 Tablet PC running Windows XP. A stylus, with its barrel buttons disabled, was used for all input.

5.2 Participants

Ten male volunteers, aged 20-26, participated in the experiment. Three participants were left-handed, and all participants controlled the stylus with their dominant hand.

5.3 Procedure

The task environment consisted of actual sketches, obtained from a previous study where users were asked to make various sketches using Windows Journal on Tablet PCs [8]. Users tapped on a green circle to begin a trial, and a red circle to end the trial. In each trial, a goal selection was highlighted green (see Figure 8).

We tested two techniques for making the selection: the lasso icon technique (*icon*), and the Handle Flag *(handle*). We omitted the lasso hotkey technique from this study because we wanted to focus our study on button-free techniques. Pilot testing also showed that it would result in the same difference from the lasso toolbar technique as Experiment 1. Thus there was little insight to be gained by repeating the technique in this experiment.

Figure 8. Experiment 2 environment. Users selected the green highlighted area from a sketch.

The *icon* technique was identical to the technique tested in the first experiment. If more than 50% of an ink stroke was contained by the lasso, the stroke was selected.

For the *handle* technique, we manually created a reasonable hierarchal grouping of the ink strokes for each sketch. This allowed us to approximate a decent parser, as our focus was not on testing our own parsing algorithm. Handle Flags were added for all of these groups, and also for the individual strokes. All of these Handle Flags were "live" during the *handle* technique trials. We then choose goal selections which would result in one of 3 possible conditions for the *handle* technique - the goal was either at the root level (*root)*, at the second level of the hierarchy (*second)*, or non-existent, requiring the user to use the integrated lasso tool to make the selection (*lasso)*. The groupings were prioritized based on the number of strokes they contained, so goal selections at the root level were larger. Also, goal selections requiring the integrated lasso tool consisted of objects which were not directly adjacent. We felt this would most closely match actual application use, where users may have some information as to how the Handle Flags were ordered and what selections existed, but would not be sure.

5.4 Experimental Design

A repeated measures within-participant design was used. The independent variables were *Technique* (*icon*, *handle*) *Level* (*root, second, lasso*), and *Block* (*1 – 4*). The study was completed in one session lasting approximately 75 minutes. The session was broken up by the two techniques. For each technique, participants progressed through the same 4 sketches, in a random order. Two of the sketches were of grocery lists, and two of the sketches were of directions to a house. Within each sketch there were 12 possible goal selections, 3 for each possible value of *Level*. This *Level* variable is most relevant to the *handle* technique, but the 12 possible selections were the exact same for the Lasso technique. For each of the sketches participants performed 4 blocks of trials. Within each block, each of the 12 possible goal selections was presented once, in random order. Because goal selections were at the same Handle Flag level in each block, users had a chance to learn and possibly remember where they were. Thus, the block stood as a variable for both overall technique learning and for the users' understanding of the underlying groupings. Presentation order of the techniques was fully counterbalanced.

Before trials began for each technique, users completed a 5 minute warm-up session, using a fifth sketch of a house which was only used for the warm-ups (Figure 8).

5.5 Results

5.5.1 Trial Completion Time

Trial completion time was defined as the time between releasing the pen after tapping the start button, and pressing down the pen on the end button. We discarded trials in which errors occurred. Due to a software error, one of the ten participant's data files was corrupted. Thus, our analysis was performed on the first 8 participants, so that the technique remained fully counterbalanced.

Repeated measures analysis of variance showed main effects for *Technique* ($F_{1,7} = 7.18$, $p < .05$), *Block* ($F_{3,21} = 6.47$, $p < .005$), and *Level* ($F_{2,14} = 378$, $p < .0001$). Average trial completion times were 5.11s for *handle* and 5.92s for *icon* (Figure 9a). However, this improvement should not be taken as a metric of "overall benefit", because of a strong *Technique X Level* interaction effect ($F_{2,14} = 89.3$, $p < .0001$), described below.

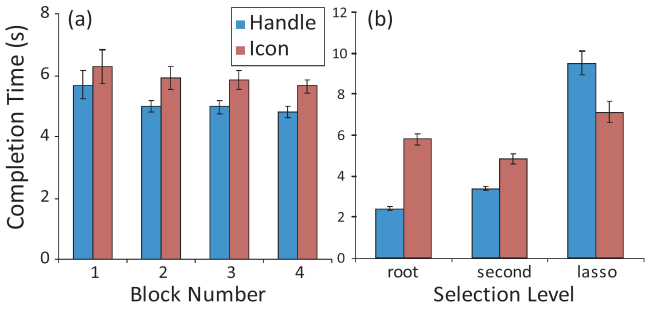

Figure 9. Completion times by (a) *Block* (b) *Level*.

The *Technique X Level* interaction is illustrated in Figure 9b. It can be seen that *Level* had a strong effect on both techniques. Even though *Level* was meant as a variable for the Handle Flag technique, the variable also had a correlation with the complexity of the required lassos. This is because the Handle Flag groups were ordered by the number of strokes they contained, so larger selections were at the top level (*Level = root*), and smaller selections were at the second level (*Level = second*). Furthermore, selections which did not have associated Handle Flags (*Level = lasso*) contained non adjacent objects, which made the selections more complex for the lasso technique as well. The *handle* technique was faster at *Level = root* (2.4s vs. 5.8s) and *Level = second* (3.4s vs. 4.8s), both at the $p < .05$ level.

It is interesting to note that these Handle Flag completion times are consistent with the results from Experiment 1, where, unlike this experiment, users knew the level of the goal Handle Flag

before the trial began. For *Level = root*, the completion times from this experiment were 0.19s longer, and for *Level = second,* completion times were 0.15s longer. In just the first block, when users would not have known where the goal Handle Flag existed, completion times were 2.6s for *Level = root* and 3.6s for *second*, which is 0.28s and 0.22s longer than in the first block of the first experiment respectively. Although this data is from two separate experiments, it does seem to indicate there will only be a minimal overhead cost introduced when users do not know where the goal handle will be located, and shows that the hierarchy is an effective mechanism for navigating to a goal selection.

In comparison, the completion times for the lasso techniques were much greater than in Experiment 1. As a result, for *Level = root* and *Level = second*, the results were *more* favorable towards Handle Flags than in the abstracted environment of Experiment 1. The overwhelming effect of using real sketches was that lassoing selections became quite difficult. Often the goal selections, which were all chosen to be reasonable and plausible selections from actual sketches, existed in dense areas of surrounding strokes, and were quite difficult to lasso. This is demonstrated in Figure 10, which shows the sorted average times taken to lasso each of the 48 goal selections used in the experiment (4 sketches x 12 goals per sketch). It can be seen that there is a large variation, and the most difficult selections were quite time consuming, with one selection taking on average more than 10 seconds.

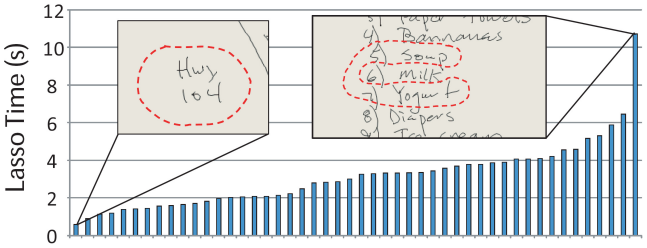

Figure 10. Average lasso times sorted across the 48 goal selections. Inset: The goal selections which were fastest and slowest to lasso.

In trials when the integrated lasso was required, *handle* was slower (9.5s vs. 7.1s). This shows that the main overhead cost associated with Handle Flags arises when a selection does not exist. We noticed from both observations and user comments, that often users wished that there was no dwell period before the integrated lasso icon faded in (which was set at 0.4s). This is a consideration for future designs.

5.6 Error Rates

Error rates were significantly affected by *Technique* ($F_{1,7} = 9.3$, $p < .05$). The total error rates were high – 18.2% for *handle*, and 32% for *icon*. The majority of errors for *handle* occurred when *Level = lasso*, in which the error rate was 39%. For *Level = root* and *Level = second* the error rates for *handle* were only 6.1% and 9.4%. For *icon* the error rates were between 25% and 35% for all values of *Level*. These high error rates further justify our motivation of providing alternatives to lassoing.

6 Application Integration and Observations

To obtain feedback on Handle Flags outside of the experimental settings, we ran 5 users through a 30 minute think-aloud usage observation session with a sketching application which implemented Handle Flags as the selection mechanism. We gave users a short briefing on Handle Flags, and then asked them to carry out a series of sketching tasks that were constructed to elicit selection hierarchies and transitions between selections. For example, in one task, we asked users to draw and then move a house and its windows, but we subsequently asked users to move only the windows. Overall, these sessions were encouraging. Users seemed to form a solid understanding of the Handle Flag hierarchy, and often knew which selections would and would not be available ahead of time. Participants used a combination of the compound selection tools (adding or removing strokes offered by Handle Flags from the current selection) and the integrated lasso tool to form new selections. Users understood that after forming new selections, they would be available though Handle Flags.

Users did make some errors with the Handle Flags. Most common was confusion about when to hover and when to tap. Users sometimes tapped on the origin of the Handle Flag, instead of the handle. To expand the hierarchy, users sometimes tapped on the handle instead of hovering. However, almost all such errors were made in the first couple of minutes of the sessions.

6.1 False Activations

An important consideration to examine is if Handle Flags will be activated by accident while users are performing other actions, such as quickly jotting down notes. We obtained the data set from a previously published study [8] of roughly 3 hours of Windows Journal pen data from 15 different users. This data was used to simulate user input in our sketching application. Any time a Handle Flag was activated indicated that a false activation would have occurred if Handle Flags were being used.

This simulation resulted in only three Handle Flag activations, or approximately one per hour of continuous pen activity. This is a very low rate of false activation; even status-quo methods such as pen buttons can suffer from high rates of accidental activation [15]. Since a false activation of a Handle Flag only brings up a menu, the cost of these infrequent errors is relatively small for the user, and in the worst case, can easily be undone.

7 Implications and Limitations

The results of our studies indicate that Handle Flags could be a useful addition to pen based interfaces. The first experiment hinted at potential quantitative benefits, but its main purpose was to gather data about the mechanics of the Handle Flag technique, so it was performed in a controlled abstract environment.

As such, we conducted a second experiment to investigate the benefits of Handle Flags in an actual application setting. In this experiment Handle Flags were significantly faster when selections were at both the root (142% improvement) and second level (41% improvement) of the hierarchy. The added cost as a result of users not knowing where the goal selections would be before each trial was minimal, in the range of 0.15s-0.3s. While the second experiment did not compare Handle Flags to a non-dominant hand button-activated lassoing mode, we can estimate the comparison based on the data from the first experiment, when the *button* technique took approximately 0.75s less than the *icon* technique. Under this assumption Handle Flags would have still been faster when selections were both at the root and second level of the hierarchy. In addition, a button-activated lasso would still suffer from the high error rates associated with the lassoing, which Handle Flags significantly reduced.

The main limitation of the Handle Flags is when a selection is not available from the Handle Flag hierarchy. In our second experiment, we found that using the integrated lasso tool was significantly slower than using the toolbar icon. However, even though 1/3 of all the trials in Experiment 2 required a lasso, Handle Flags were still significantly faster overall. Furthermore, Handle Flags are not meant to replace the lasso tool, but to complement it. In an actual application, the user could access the lasso icon from the toolbar when required.

That being said an important consideration is that the need to choose whether to use the lasso or Handle Flag could introduce a cost in itself. However, our usage observation sessions indicated that when users have created the sketches themselves, they will form a cognitive model of which strokes are available by Handle Flags, so such a cost may be minimal.

8 Future Work

Our results indicate that alternatives to lassoing should be considered for ink applications, as lassoing is time consuming and error prone. Handle Flags are only one such option. Another alternative is to tap to cycle possible selections [26]. Previous implementations of tapping were developed for mouse-based image editors, where commands were accessed using a right button click. We have started to explore the integration of the tapping mechanism into Handle Flags, as an alternative to displaying and navigating the hierarchy (Figure 11). In the future, we hope to evaluate this approach in comparison to navigating the actual hierarchy. It is unclear if tapping would be a more efficient technique, because it forces users to do a linear search through all potential selections.

Figure 11. Tapping the icon in the top left corner of the Handle Flag cycles through possible selections.

In our implementation, the prioritization order of the Handle Flags was entirely determined by recency of use. However, if users are frequently changing selection scope, it may make the location of the desired Handle Flag unpredictable. This could outweigh the benefit of having the most recent selection at the root level. It would be useful to explore other possible heuristics for ordering selections. For example, in our second experiment, we found that ordering the selections by the number of strokes contained within allowed users to quickly find the goal selection.

One final issue that should be explored is the implicit grouping algorithm implemented by the system. In our application, these implicit groupings were formed using a simple heuristic based on spatial properties of the ink strokes. While informal usage observation of our sketching application showed that even coarse recognitions can enable the use of Handle Flags, it would be interesting to combine Handle Flags with state-of-the-art sketch recognition or more sophisticated ink parsing systems. A system which recognizes not only groups, but also hierarchies of groups, would have a natural integration with Handle Flags, since Handle Flags allow users to select from these hierarchies. For example, if the user wrote a sentence, the system could recognize the individual strokes, the characters, the words, and the entire phrase.

References

[1] Accot, J. and Zhai, S. (1997). Beyond Fitts' Law: Models for trajectory-based HCI tasks. *ACM CHI*. 295-302.

[2] Accot, J. and Zhai, S. (2002) More than dotting the i's - foundations for crossing-based interfaces. *ACM CHI*. 73-80.

[3] Alvarado, C. (2004). Sketch Recognition User Interfaces: Guidelines for Design and Development. *AAAI Fall Symposium on Intelligent Pen-based Interfaces*. 8-14.

[4] Buxton, W., Fiume, E., Hill, R., Lee, A. and Woo, C. (1983). Continuous Hand-Gesture Driven Input. *Graphic Interface*. 191-195.

[5] Deming, K. and Lank, E. (2004). Managing Ambiguous Intention in Mode Inferencing. *Proc of the AAAI Fall Symposium Series*. 49 - 54.

[6] Fitzmaurice, G., Balakrishnan, R. and Kurtenbach, G. (1999). An exploration into supporting artwork orientation in the user interface. *ACM CHI*.167-174.

[7] Fitzmaurice, G., Khan, A., Pieke, R., Buxton, B. and Kurtenbach, G. (2003). Tracking menus. *ACM UIST*. 71-79.

[8] Grossman, T., Hinckley, K., Baudisch, P., Agrawala, M. and Balakrishnan, R. (2006). Hover widgets: using the tracking state to extend the capabilities of pen-operated devices. *ACM CHI*. 861-870.

[9] Guiard, Y. (1987). Asymmetric division of labor in human skilled bimanual action: The kinematic chain as a model. *Journal of Motor Behavior*. 19(4): 486-517.

[10] Hinckley, K., Baudisch, P., Ramos, G. and Guimbretiere, F. (2005). Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. *ACM CHI*. 451-460.

[11] Hinckley, K., Guimbretiere, F., Agrawala, M., Apitz, G. and Chen, N. (2006). Phrasing techniques for multi-stroke selection gestures. *Graphic Interface*. 147-154.

[12] Hinckley, K., Guimbretiere, F., Baudisch, P., Sarin, R., Agrawala, M. and Cutrell, E. (2006). The springboard: multiple modes in one spring-loaded control. *ACM CHI*. 181-190.

[13] Kurtenbach, G. and Buxton, W. (1991). Issues in combining marking and direct manipulation techniques. *ACM UIST*.137-144.

[14] Lank, E. and Saund, E. (2005). Sloppy Selection: Providing an Accurate Interpretation of Imprecise Stylus Selection Gestures. *Computers and Graphics*. 29(4): 490-500.

[15] Li, Y., Hinckley, K., Guan, Z. and Landay, J. A. (2005). Experimental analysis of mode switching techniques in pen-based user interfaces. *ACM CHI*. 461-470.

[16] Mankoff, J., Hudson, S. and Abowd, G. (2000). Interaction techniques for ambiguity resolution in recognition based interfaces. *ACM UIST*. 11-20.

[17] Mizobuchi, S. and Yasumura, M. (2004). Tapping vs. circling selections on pen-based devices: evidence for different performance-shaping factors. *ACM CHI*. 607-614.

[18] Moran, T. P., Chiu, P., Melle, W. v. and Kurtenbach, G. (1997). Pen-based interaction techniques for organizing material on an electronic whiteboard. *ACM UIST*. 127-136.

[19] Mynatt, E., Igarashi, T., Edwards, W. and LaMarca, A. (1999). Flatland: New dimensions in office whiteboards. *ACM CHI*. 346-353.

[20] Norman, D. A. (1981). Categorization of Action Slips. *Psychology Review*. 88(1): 1-15.

[21] Plamondon, R. and Srihari, S. (2000). On-line and Offline Handwriting Recognition: A Comprehensive Survey. *IEEE Pattern Analysis*. 22(1): 63-84.

[22] Ramos, G., Robertson, G., Czerwinski, M., Tan, D., Baudisch, P., Hinckley, K. and Agrawala, M. (2006). Tumble! Splat! helping users access and manipulate occluded content in 2D drawings. *AVI*. 428-435.

[23] Ren, X. and Moriya, S. (2000). Improving selection performance on pen-based systems: a study of pen-based interaction for selection tasks. *ACM TOCHI*. 7(3): 384-416.

[24] Saund, E., Fleet, D., Larner, D. and Mahoney, J. (2003). Perceptually-supported image editing of text and graphics. *ACM UIST*. 183-192.

[25] Saund, E. and Lank, E. (2003). Stylus input and editing without prior selection of mode. *ACM UIST*. 213-216.

[26] Saund, E., Mahoney, J., Fleet, D., Larner, D. and Lank, E. (2002). Perceptual Organization as a Foundation for Intelligent Sketch Editing. *AAAI Symposium on Sketch Understanding*. 118-125.

[27] Saund, E. and Moran, T. P. (1994). A perceptually-supported sketch editor. *ACM UIST*. 175-184.

[28] Sellen, A., Kurtenbach, G. and Buxton, W. (1992). The prevention of mode errors through sensory feedback. *Human Computer Interaction*. 7(2): 141-164.

[29] Zeleznik, R. and Miller, T. (2006). Fluid inking: augmenting the medium of free-form inking with gestures. Graphics Interface. 155-162.