Fluid Inking: Modeless Integration of Gestures with Inking

Robert Zeleznik and Timothy Miller

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-05-10
July 2005

Fluid Inking: Modeless Integration of Gestures with Inking

Robert Zeleznik and Timothy Miller
Computer Science Department
Box 1910
Brown University
Providence, RI 02912
{bcz,tsm}@cs.brown.edu

ABSTRACT
We present Fluid Inking, a novel approach for integrating gestural commands with freestyle inking that is also more generally applicable than previous techniques. Fluid Inking adapts the conventional GUI notions of tear-off menus and modifier-key shortcuts into the novel concepts of tear-off FluidButtons and gestural shortcuts, respectively. Fluid Inking is distinct among gesture-based interfaces because it fundamentally addresses four cornerstone gestural design criteria, including: 1) hardware generality: all gestures, including menu interactions, are available using only a button-free stylus, the least common denominator input device for systems ranging from PDAs to whiteboards; 2) performability: gestures are composed of very short sequences of simple and familiar inking interactions that require minimal targeting; 3) extensibility: gestures are conceived of as shortcuts for many of the commands in an arbitrarily scalable FlowMenu; and 4) discoverability: gesture shortcuts are displayed in the interactive FlowMenu and, in novice mode, they are suggested with dynamic feedback during inking. This paper presents the Fluid Inking design in the context of a broad prototype notetaking application with emphasis on how we use context, postfix *terminal punctuation*, and prefix *Flicks* to distinguish gestures from regular inking. We also discuss how the Fluid Inking design evolved based on user feedback.

ACM Classification H.5.2 User Interfaces—Graphical user interfaces, H.5.2 User Interfaces—Input devices and strategies, H.5.2 User Interfaces—Interaction styles

General Terms Design, Human Factors

KEYWORDS: gestures, modeless, terminal punctuation, tablet computing, FlowMenu

1 INTRODUCTION

Our focus in this work was to address *all* four of the cornerstone design requirements that we believe are necessary to achieve a fluid user interface, including: hardware generality, performability, extensibility, and discoverability. Since the two prevailing and previously unintegrated gestural interface approaches—memorizing a catalog of "invented" gesture symbols, and automatizing marking menu interactions—are each successful with regard to some of these requirements, our solution blends a hybrid of their best features with some novel concepts.

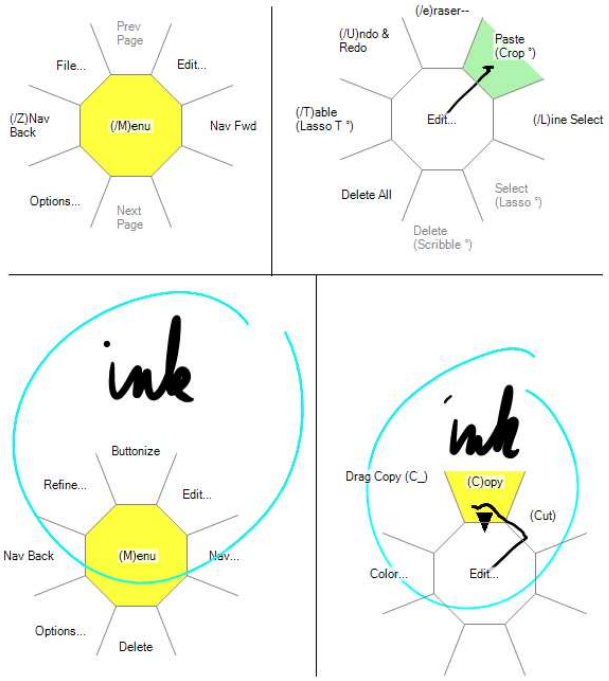

Figure 1: A FlowMenu sequence. A 500ms press-and-hold raises the menu. Paste is selected. A 200ms press-and-hold inside the pasted selection raises a local menu. Copy is then chosen. Shortcuts are noted in the menu with ()'s.

Despite the success some people have with special buttons, achieving hardware generality was of paramount importance to us. Specifically, at least for some people, some of the time, it is not feasible due to lack of universal availability or not acceptable because of individual preferences to *require* anything other than a button-free stylus. Problems commonly arising with integrated stylus buttons include: errors caused

Figure 2: The same action sequence as Figure1 but done with shortcuts. An L-shaped crop stroke is drawn to specify the lower-left corner of the pasted ink. A tap inside the crop's hull pastes ink. A 'C' overlapping the selection copies the ink. A 200ms pause at the end of C stroke pastes (i.e., duplicates) and drags the copy.

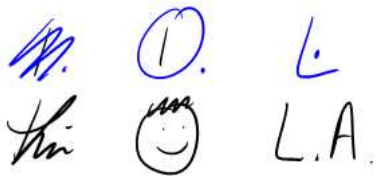

Figure 3: The blue strokes in the top row each correspond to two-stroke gestures (delete, select, and paste), assuming the tap comes last. Despite superficial similarity to those on top, none of the strokes in the second row triggered gestures. The smiley face, however, might trigger a gesture if drawn in an unusual order (e.g., mouth, head, eyes or face, hair, eyes).

by accidental or failed triggering [11], finger strain, and awkwardness. External buttons also can be awkward, and are not appropriate in common situations where the second hand is better served doing something else, like notetaking while holding a phone or a book. These observations do not imply a conclusion that special hardware should be avoided, but rather that it should be optional and perhaps complementary. For instance, available special buttons could provide direct access to additional functionality or could suppress gesture recognition in ambiguous situations such as when drawing the Fluid Inking gesture set itself. See Figure 3.

We addressed our performability requirement by using multi-stroke gestures comprised of sequences of simple, familiar strokes instead of the more familiar chunking of an entire cognitive action in a single physical phrase. Despite benefiting from mode avoidance, chunking can create performance problems [19]. Our primary performability contribution lies in the novel techniques we developed for distinguishing gestures from regular inking using prefix *Flicks* [14] (fast straight lines, in our system diagonally up to the right) and postfix *terminal punctuation* (either fast taps, or pauses).

Our extensibility requirement is straightforward to meet when regarded independently, particularly by using hierarchical menu systems. However, *effective* extensibility, which considers collectively our other design requirements, is more challenging. Our hybrid approach extends the capabilities of hierarchical menus with a catalog of gesture shortcuts that can directly index deeply nested functions. Any physical automatization of these shortcuts is not disrupted if the menu structure ever changes, for instance to accommodate new commands—a fundamental problem with derivatives of marking menus. We also adapted the classical notion of tear-off menus for use within FlowMenus and in a more novel way for gesture shortcuts. The resulting torn-off elements—functionally enhanced pushbuttons called FluidButtons—provide extensible and direct ad hoc function access.

Our discoverability criterion, when individually considered, is also easily met by marking menu techniques which are intrinsically browsable. Although we did attempt to make FlowMenu interaction more self-disclosing with explicit visual feedback, our primary concern was to facilitate the discovery of our integrated catalog of shortcut gestures. Borrowing again from traditional menus that indicate modifier-keys, we enhanced FlowMenus with similar notations for our shortcuts. In addition, while inking, we supply dynamic feedback, either verbose to direct the novice, or subtle to provide recognition confirmation for the trained user.

The body of this paper details how Fluid Inking addresses our four design criteria by presenting our primary contributions in three integrated areas, including:

- *FlowMenu enhancements*: tear-offs; shortcuts; button-free invocation; and active feedback.
- *Multi-stroke gesture shortcuts*: Mnemonic Flicks, terminal punctuation, contextual gestures, and drawing tear-off buttons; a consistent, broadly functional gesture catalog; and verbose and subdued feedback.
- *Gestural widgets*: enhanced pushbuttons, called FluidButtons; and a gestural direct manipulation widget.

2 RELATED WORK

We presented the preliminary notions of a punctuated gesture interface as part of our MathPad^2 system [10]. MathPad^2's gestures omit most of the functionality of this work by focusing only on narrow non-systematic use of tapping.

FlowMenus [4], marking menus [8], and Scriboli [5] are related gesture techniques that share a common notion of a radial menu. Each of these systems is compatible with Fluid Inking and could be used as a backbone for discovering our gesture set. In each case, we perceive the value of our gestures as being alternative and potentially simpler and more natural "shortcuts" for menu operations. In addition, we note that to our knowledge none of these designs have considered interaction using *only* a button-free stylus. Our enhanced FlowMenu design not only supports such a restricted input device but also provides abbreviated interactions for triggering first-level menu items.

Moran, et al. [13] presented a suite of gesture-based interaction techniques targeting collaborative electronic whiteboard interaction, many of which are closely related in spirit to ours. Yet, because of their *openness principle*, which does not tolerate restrictions on the kinds of ink marks that can be drawn (i.e., by pre-defining certain types of ink marks as gestures), they made only an abortive attempt at using a postfix indicator (terminal punctuation) to avoid the mode switch between inking and gesturing [16]. They encountered difficulties when trying to distinguish their double-tap punctuation from regular "choppy" handwriting and abandoned the technique, in favor of pen modes which, as they report, come

at the "cost of users having to be vigilant." Their experience notwithstanding, we designed a full gesture set using different punctuation that, in practice, is generally unaffected by violations of the openness principle. We also address problems they noted with the learnability of a large gesture set with our hybrid FlowMenu design.

Alvarado, et al. [1] presented a modeless gestural user interface as part of their ASSIST system for creating and understanding mechanical sketches. Their interface does not require punctuation because their domain-specific drawing restrictions (i.e., mechanical objects) allow gestures to coexist with inking. Yet even with such restrictions, they require user assistance to override interpretation errors that we believe could generally be avoided with gesture punctuation.

Many other systems combine gestural interpretation with free-form annotation, such as Landay's SILK [9] and Mynatt's Flatland [15]; however, they use explicit prior actions (e.g., button press) to disambiguate gestures from notes.

Saund and Lank [17], however, present a technique for avoiding prior selection of mode during inking, that is related both to our terminal punctuation gestures and to the general notion of suggestive interfaces developed by Igarashi [6]. However, their technique only avoids prior mode selection for the sub-task of forming a selection. Additionally, their "select" button may interfere with subsequent drawing or can require fine-grained targeting to select.

Mankoff, et al. [12] consider general techniques for resolving ambiguity in recognition-based interfaces. However, our desire for modeless drawing and gesturing guarantees that theoretically all gestures will be ambiguous with possible notations. Thus, an explicit mediation-based resolution system would not be appropriate because it would necessarily disturb interaction flow. Instead, our work focuses on the complementary problem of how to design gestures that will not, in practice, require ambiguity resolution.

Buxton [3] noted the value of making a gesture match the mental model of its action to unify a command and arguments in a single "chunk" that avoids errors of syntax. Although this has been one of many important criteria in our design, we believe strict adherence to his uni-stroke "phrasing" can be physically awkward, abstract, and potentially hard to segment precisely [7] (compared to naturally segmented multi-stroke gestures). Additionally, previous evaluations have shown relative advantages for multi-stroke gestures; for instance, Zhao and Balakrishnan [19] found that a hierarchical marking menu selected via a series of simple lines worked better than the original technique using a single zig-zag.

3 FLOWMENU ENHANCEMENTS

The FlowMenu [4] is a radial menu invoked with either a stylus-mounted button or an external button held in the non-dominant hand. The menu is always visible during interaction and supports hierarchical navigation via backward and forward crossings between the menu center and radially-placed items.

We adapted the invocation of FlowMenus to stylus-only use

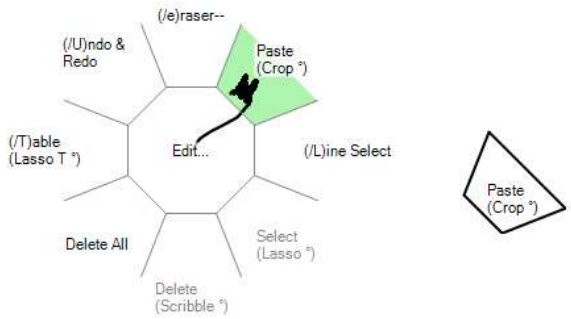

Figure 4: Except for Delete All, all Edit menu items have a gestural shortcut noted with ()'s. / stands for a Flick, *o* stands for tap or pause, and _ for pause (not seen in figure). The torn-off button was created with a wiggling action in the paste octant.

by borrowing from common PDA-based interactions in which menus are raised with a press-and-hold technique. In our implementation, we display a FlowMenu if the user presses the stylus on the display surface and holds for 500ms, consistent with the minimal acceptable timings found by Hinckley, et al. [5]. However, timeout length could be adjusted to match personal preference since some people are more likely than others to pause at the beginning of an ink stroke. Despite reduced hardware requirements and compatibility with prevalent PDA interfaces, our press-and-hold technique is disadvantaged relative to the original, instantly-appearing, button-based FlowMenu invocation. Therefore, we further enhanced FlowMenus to support gestural shortcuts and tear-off actions that can raise the menu instantly, or, for many actions, eliminate the need to use it at all.

3.1 Gestural Shortcuts

Gestural shortcuts for menu items and submenus are displayed in the FlowMenu (Figure 4) similarly to the way modifier-key shortcuts are shown in traditional menus. However, instead of "ctrl-" and "alt-" abbreviations, we use "/" and "○" to indicate *Flicks* (Figure 5) or *terminal punctuation* (Figure 3). Just as with control and alt, variant forms of punctuation not only can increase the space of potential shortcuts, but also can provide a semantic distinction within that space. In our implementation, we use Flicks followed by a mnemonic word or letter to perform meta-functions like displaying a menu or widget (e.g., Flick-U brings up an Undo/Redo FluidButton), and we use terminal punctuation, preceded by a multi- or single-stroke drawing, for actions that directly operate on a spatial selection (Figure 3). These shortcuts are discussed in greater detail in Section 4.

3.2 Tear-off Shortcuts

We allow menu items to be torn off (i.e., copied) as persistent FluidButtons in order to provide more direct access, to facilitate repetitive actions, to avoid hand occlusion and tension when inspecting a menu, and to reduce memorization requirements for shortcuts. Wiggling the pen back and forth while navigating a FlowMenu, as if trying to dislodge something from the pen tip (Figure 4), will tear off the nearest menu item. We recognize this wiggle motion by detecting more than 6 cusps (sharp direction turns) occurring within

Gesture Class	Context	Gesture	Terminal
Mnemonic Flicks	flick	symbols	—
Punctuated:			
Self-contained	—	, , or	tap/pause
Parametric:			
mnemonic	lasso	symbols	tap/pause
mimetic	lasso	action	tap/pause
Contextual	(active selection)	any	—/pause

Table 1: Classification of gesture shortcuts based on structure of gesture sequence. Symbols include letters, lassos (), crops (e.g.,) and scribbles (). Actions are imitated interactions; stroke hooks (the only actions used) imitate interactions with a flowmenu.

a bounding box of less than 50 pixels on a side in less than half a second. The cusps additionally must be in the form of a tight zig-zag, so that each must be more than 50% as far away from its neighbors as they are from each other. When a tear-off event occurs, the menu item becomes a FluidButton and is dragged with continued stylus movement and later can be manipulated or deleted in the same manner as an ink stroke.

A torn-off menu or submenu, represented as a FluidButton, can be activated instantly with a stylus press. If the stylus lifts immediately (i.e., a click) then the menu remains active, permitting deliberate menu inspection and allowing further interaction with a second stroke that implicitly begins at the menu center. Alternatively, a torn off menu items becomes a FluidButton that is functionally very similar to the menu item, but generally has greater expressive power. See Section 5.

3.3 Radial Interactions

We also adapted for FlowMenus the simple, conventional radial menu technique of direct item selection without returning the stylus to the center menu region. This interaction is only permitted after crossing *directly* from the menu center to an item and releasing. The FlowMenu can still be cancelled by crossing into any other octant before releasing. We highlight menu items in yellow when they are in focus and in green if releasing will trigger them. We also provide an arrow highlight that indicates the direction to go to complete a menu selection (i.e., toward the center). This highlighting is useful for revealing the hybrid radial menu/FlowMenu behavior.

4 MULTI-STROKE GESTURE SHORTCUTS

Gesture shortcuts are an alternate way to invoke functions. The FlowMenu is not used for invoking these shortcuts, but can be used to discover them. Similarly, shortcuts do not require an initial pause because their carefully chosen definitions of ink stroke sequences are modelessly distinguishable from regular ink. That is, ink strokes only become gesture tokens if they are part of an incoming stroke sequence that matches a complete gesture definition. Once matched, the strokes are deleted and the gesture's function is fired. Although in theory these definitions are ambiguous with general inking, in typical practice gestures are distinct because

Figure 5: A potential Flick, a Mnemonic Flick, and a non-Flick. The 'cents' sign is not a Flick because the 'c' was drawn before the line, and the line was drawn from top to bottom and too slowly.

of punctuation, stroke ordering, and stroke dynamics (Figures 3 and 5). In the infrequent cases where they are not distinguishable, undo can be used to recover from an accidental recognition, and either undo or delete for failed recognition. Alternatively, gesture recognition as a whole can be enabled or disabled from the FlowMenu (press-and-hold and Flick-M are never disabled to preserve menu access).

Besides being distinguishable from ink, our punctuated gesture sequences are notable for using only a small set of broadly-defined and familiar tokens. In fact, different tokens can even have overlapping geometrical definitions and still be distinguished by their context in a gesture sequence (e.g., context disambiguates the letter 'O' token from a lasso.) This contrasts with unistroke gesture catalogs where every gesture stroke must be geometrically distinct from all others and perhaps from regular inking as well.

Based on the structure of gesture sequences, we distinguish three classifications of shortcuts (Table 1): Mnemonic Flicks that have a prefix Flick; terminal punctuation gestures that have a postfix tap or pause; and contextual gestures that need a spatial mode (i.e., active selection or widget).

4.1 Prefix Flick punctuation

Based on the positive results Moyle, et al. [14] demonstrated when using flicks to signal an out-of-band action while the stylus (or mouse) is otherwise in a text selection mode, we were eager to explore how they might be adapted for use when the stylus was in an inking mode. Naive application of flicking will not work because of frequent ambiguities, such as trying to distinguish a flick from a quickly drawn number '1'. However, by informally surveying a variety of drawing and writing styles, we found that flicks could be disambiguated from regular inking if they were immediately followed by an overlapping mnemonic character(s). These Mnemonic Flicks increase the number (and perhaps memorability) of functions that can be indicated with Flicks (Figure 5). After initial prototyping, we determined that it was preferable to recognize only flicks drawn from bottom-left to upper-right; allowing other flick directions generated conflicts with regular handwriting and, if the directions were distinguished from each other, complicated the concept. In addition, commmon handwriting ambiguities, for example between '2' and 'Z' or the first stroke of 'T' and the single stroke of '1', prevented simultaneous use of all possible characters and sequences as mnemonics. Instead, for instance, we treat recognition results of '2' or 'Z' as the same mnemonic.

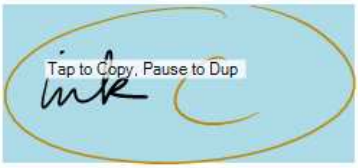

Figure 6: A group of strokes can be copied with a parameterized gesture. First a lasso is drawn around the strokes, then a 'C' is drawn inside the lasso. If a tap is drawn the strokes inside the lasso will be copied. If a press-and-hold is done, the strokes will be copied, selected and ready to be interactively dragged.

4.1.1 Mapping Flick functionality. Even though prefix Flicks are capable of specifying a 2D rectangle, their speed requirement makes them best suited for specifying at most a single point on the screen (the start of the Flick). Therefore, we only map Flicks to functions requiring no spatial context other than perhaps a location (e.g., to place a widget.) Representative examples of Flick-based functions are Flick-M to display the main menu, Flick-S to show a save dialog, Flick-U to display an Undo FluidButton, Flick-e to change the stylus to an interactive eraser mode, and Flick-L to change the stylus to an interactive line intersection selection mode. In the latter two cases, we experimented with transient modes that revert to inking after the next stroke, and lasting modes that revert after an explicit tap. Both of these are useful so the default behavior is a user preference.

4.2 Terminal Punctuation

We conceived of terminally punctuated gestures based on our theory that the least cognitively disruptive way to signal a gesture during inking would be to first "say what you want to do" by drawing all of the gesture parameters, and only then indicate to the system that what was drawn should in fact be interpreted as a gesture. Creating such a terminally punctuated gesture set presents two challenges in terms of our general design requirements, specifically how to distinguish punctuated gestures from regular inking without special hardware, and how to provide feedback of the state of the system given that until punctuation is specified, the preceding input may or may not be one of many gestures. We will discuss feedback Section 4.5.

Our solution to gesture/ink disambiguation is framed around the belief that there are many possible sequences of simple ink markings that are in practice rarely used. In particular, tapping is a very frequent activity, used, for example, to make a period, a ':' or the letter 'i.' However, outside of such restricted contexts, tapping is less common. Similarly, pausing with the stylus pressed to the screen during a stroke is not a pervasive inking activity. We therefore chose these two actions to serve as variant forms of terminal punctuation, with the caveat that careful consideration is required to insure the relative distinctiveness of collective gesture sequences from regular inking.

4.2.1 Mapping Terminal Punctuation Functionality. Just as the physical implications of keyboard layouts influences modifier-key mappings (i.e., cut, copy, paste modifier key shortcuts are all located on left-hand side of the keyboard to facilitate keeping the right hand on the mouse), the physical nature of gesture sequences influences gesture mappings. Since tapping can be performed very quickly but ends abruptly with the stylus off the surface of the tablet and not necessarily near the tap point, it is best suited to immediately performing a complete, non-interactive action. Alternatively, pausing is by definition slower, but ends with the stylus on the tablet surface at a very controlled location; it is thus well suited for signalling a transition to a transient interactive mode that ends when the stylus finally leaves the surface. However, the 500ms pause duration also implies a deliberateness that some people find unsuitable for interactive tasks since it slows down the pace of interaction. Therefore in general, we think of pause as being best suited for entering infrequently used interaction modes, for triggering interactive assistance, or for enabling simple, redundant access to interactive functionality. The pause modes, notably, come at little cost, since users need only learn the base form of gestures and a single punctuation rule to command the full gesture set.

Figure 7: Parameterized gesture variants for 'Cut'. On the left, a Mnemonic gesture using the common 'X' for cut. On the right, a mimetic gesture that represents the FlowMenu action for accessing the Cut menu item.

Terminally punctuated gestures also can be divided logically into *self-contained* and *parameterized* groups based on their gesture sequence structure (Table 1). The class of *self-contained* gestures are those that consist of one gesture token that specifies both an action and its spatial context followed by punctuation (Figure 3). The *parameterized* gesture class consists of sequences containing one token to indicate a spatial context, followed by a mnemonic or mimetic token to indicate an action, followed by punctuation (Figures 6 and 7).

4.2.2 Self-contained gestures. Examples of self-contained gestures are the paste, select and delete gestures. The paste gesture is indicated by drawing a crop mark (e.g., ⌞) that specifies where to place the corresponding bounding box corner of the clipboard contents and then tapping or pausing within the crop's convex hull. The crop mark itself is sufficient to indicate both the paste function and all of its spatial parameters, but punctuation is needed to distinguish it from regular inking. A tap to paste the clipboard contents must be done inside the crop's convex hull to avoid conflicts with general inking; for instance, writing the number "7.0" would otherwise be misinterpreted as a paste gesture followed by the number '0'. Pause punctuation engages an interactive menu of paste buffers, a la Microsoft Visual Studio's clipboard ring.

The select gesture is indicated by drawing a lasso around a group of objects. The lasso definition is very broad and roughly includes any curve that travels at least 25% of its arc length away from its starting point and then returns to within 25% of its arc length from the starting point. Additionally, the definition requires that the lasso stroke contain at least

Figure 8: The self-contained gesture for delete. Tapping inside the scribble will delete the contained tick marks. A tap outside the scribble will delete the axis as well. A press and hold inside the scribble makes the stylus act as an interactive eraser until lifted. Anything else leaves the scribble uninterpreted.

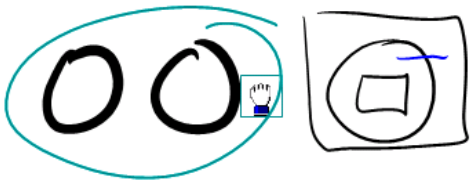

Figure 9: A line intersection selection refinement gesture (horizontal blue line) adds just the intersected nested circle to the active selection. A final tap in the selection will complete the gesture.

one stroke or object. Although in general tap punctuation should be done inside a gesture's convex hull, we relax this rule for selection since arbitrary tap locations rarely conflict with general inking. Pause punctuation selects the strokes and then raises an interactive FlowMenu.

The delete gesture is the most complicated gesture to recognize in our system because of potential ambiguities with cursive handwriting, although to the user it is simply presented as a back and forth scribble over one or more strokes. The recognition details of this gesture are available elsewhere [18], but can be summarized as requiring at least 3 cusps that must contain, using a special coverage metric, at least one stroke. Tapping performs either a narrow delete (just strokes contained in the scribble) if the tap is inside the scribble, or a broad delete (all strokes overlapping the scribble) if the tap falls outside. Pausing performs a narrow delete and initiates a transient interactive eraser mode lasting until the stylus is lifted. See Figure 8.

4.2.3 Parameterized gestures. Parameterized gestures, indicated with either mnemonic or mimetic marks, apply to actions on selections (Figure 7). A representative example of a mnemonic gesture is Copy. After drawing a lasso around a selection of strokes, they can be copied by writing an overlapping mnemonic 'C', followed by punctuation. Tap punctuation copies the strokes. Pause punctuation duplicates them—that is, copies, pastes and initiates a transient interactive drag mode until the stylus lifts. See Figure 6. Unlike modifier-key mnemonics, the ambiguity of whether 'C' stands for cut or copy can be avoided with longer mnemonics, like 'Cut' and 'Copy'. We make both short, 'C' and 'X', and long, 'Copy' and 'Cut', mnemonics available. Selections do not always have to be of strokes or objects; for instance, canvas regions can be selected for magnification. To enlarge a specific region, draw a lasso around it that may or may not contain strokes, then draw a 'Z' inside. Tapping zooms so that the region fills the window, while pausing additionally initiates a transient interactive zoom mode. Some mnemonic gestures make no special use of pausing, such as 'Col' which always displays an interactive color picker, and 'D' which always performs word recognition and then presents the results of a dictionary lookup.

A mimetic gesture is one that imitates an action. Our system currently has only indirect mimetic gestures, reminiscent of marking menus, that mimic interacting with a FlowMenu without having to actually see it. After drawing a lasso, a first level menu item can be invoked by a mimetic *stroke hook* (i.e., a straight line that doubles back on itself), followed by punctuation. For example, 'cut' lies in the northeast octant of the FlowMenu and can be invoked with a stroke hook up to the right (the same action that would be used if the FlowMenu were displayed) that intersects the selection lasso. See Figure 7. If the final punctuation is a tap, the selection is just cut, but if it is a pause, the FlowMenu appears, allowing interactive selection of the function to invoke. If the gesture lasso is empty, the global FlowMenu is used.

4.3 Contextual gestures

The primary motivation for using punctuated gestures is to distinguish them from inking. However, in a rich notetaking application, there will be specialized objects that accept gestural interaction and either no or restricted inking. For these spatial modes, we can define gestures that require no punctuation. For example, the selection object that is created when strokes are selected defines a spatial mode around these strokes. This spatial mode supports punctuated or unpunctuated gestures that are derived from the previously described parameterized gestures.

To avoid having to learn an entirely new set of gestures for the active selection, all parameterized gestures are available subject to the the regular change of omitting the initial lasso and any final tap. Pausing is optional but the pause timeout is shortened from 500ms to 200ms, making pause interactions more attractive since they feel more like a stop than a pause. If gestures are forgotten, the self-contained technique for bringing up a menu, press-and-hold, can be used with the shortened 200ms timeout.

4.3.1 Selection refinement. In addition to re-using self-contained gestures, active selections support additional gestures for refining the selection by adding, removing or toggling elements of the selection, similar to GEdit [7]. We support these additional functions by both repurposing existing gestures and adding two new gestures. In this context, scribbling deselects strokes and a lasso that overlaps the active selection adds strokes, but neither requires punctuation. However, if the lasso is disjoint from the selection, a terminal tap is needed. If the tap is inside the active selection, then the lasso contents are added to the selection; otherwise, the lasso contents become the active selection. See Figure 11.

A straight line intersection gesture (Figure 9) for toggling the selection state of strokes is also available when there is an active selection. With this gesture, the user draws one or

Figure 10: A manipulation widget is placed where the tap was made when the strokes on the left were selected using a Lasso and Tap gesture. By pressing the stylus outside the button and dragging in a straight line through it, the strokes are scaled about the corner opposite the handle. Pressing the stylus inside the button would have dragged the strokes.

more selection lines through objects to toggle whether they are selected or not. If the lines overlap the active selection punctuation is unneeded, otherwise a tap in the active selection terminates the gesture and updates the selection.

4.3.2 Selection manipulation. A primary reason for selecting objects is to translate or scale them interactively. We had originally provided only tap-based non-interactive move/resize gestures and their pause-based interactive counterparts. Unfortunately, neither of these was satisfactory since pausing was disruptive and non-interactive tapping did not afford precise positioning. Our current interface addresses these problems by adding a manipulation widget to the active selection interface.

When a self-contained select (or paste) gesture is made, an interaction widget is instantiated at the tap (or pause) location. Similar to Apitz, et al. [2], this widget leverages crossings, however, we use them to make an additional scaling mode available on an otherwise traditional (i.e., press and drag) move handle. To choose scaling, the stylus must start outside the widget and cross into it along an approximately straight line. When a crossing is detected, the selection begins interactive scaling about the corner opposite the widget (Figure 10). (A menu option is provided for setting an arbitrary center of scale within the selection.) During manipulation, the widget is hidden so as to avoid obscuration. The widget is 30 pixels square, which is large enough to support quick targeting with a pen. However, we noticed that users wanting to quickly move an object would perform the lasso and tap gesture followed immediately by a drag stroke. They would then often miss the manipulation widget target even though it was located right where their stylus had been at the end of the tap. This problem arose because they coarticulated the beginning of their drag motion with the end of their tap stroke as the stylus was lifting off the display. By the time the stylus contacted the display again for the drag, it had already moved some distance making it difficult to reflexively target the widget. We addressed this "recognition" error by doubling the pick area of the widget for the first 150ms after the selection tap.

A second potential conflict with the manipulation widget occurs when performing a gesture on the selection object that accidentally starts on the manipulation widget (causing a drag) or passes through the widget along an initial straight line (causing a scale). Making sure the handle is out of the way by initially tapping away from the selection avoids this problem as does removing the selection handle (by scribbling over it or drawing a circle anywhere outside the active selection). The handle can be restored or relocated by drawing a circle within the selection that either does not contain any strokes or ends with a pause to allow interactive positioning.

4.4 Tear-off Button Drawing

As convenient as gestures are for rapidly invoking actions, there are still occasions when a persistent button-based interface is more suitable, including when shortcuts do not exist or are hard to remember, or when an action needs to be repeated frequently. We therefore extended the notion of tear-off menus to apply also to gestures. Instead of a literal tear-off action, buttons are created by drawing first a gesture without either its prefix flick or postfix punctuation. Then the incomplete gesture strokes can be selected and converted to a FluidButton with a "Tear-off" menu command. The torn-off FluidButton is sized according to the drawn gesture strokes and corresponds in functionality to the gesture action. For example, a delete FluidButton can be created by drawing a scribble (the delete gesture sans terminal punctuation), lassoing it, and then interactively or mimetically choosing the tear-off menu item. A FluidButton for undo can likewise be created by drawing the mnemonic part of the Flick-U shortcut, 'U'. See Figure 12. Thus, related to Landay's techniques [9], a button-based interface can be laid out and customized as rapidly as drawing and manipulating mnemonic ink strokes.

4.5 Speculative Feedback

Although we expect that many users will discover gesture shortcuts by navigating through FlowMenus, we wanted a suggestive mechanism that could adventitiously direct novices to potential shortcuts. We resolved upon two feedback mechanisms, one verbose technique targeted narrowly toward true novices, and another more subdued approach with broader aims.

4.5.1 Verbose feedback As strokes are drawn on the screen, we provide a verbose mechanism that illustrates which gestures have been partially completed. For example, if a lasso is drawn around some strokes, it will be semi-transparently filled to indicate that a tap will confirm it as a selection. We additionally present text prompts in that region describing the potential gestures that start with a lasso. In some cases, more than one tap region might exist such as when a selection is active and a lasso is drawn around strokes not in the selection. See Figure 11. In this instance, the active selection will be highlighted and prompts displayed indicating that a tap inside will extend the selection, and the lasso will be highlighted to indicate other options.

This active feedback is particularly appropriate for our gesture set since our gestures consist of common inking actions. Thus it is highly likely that users will become accidentally aware of possible gestures merely by drawing. On the other hand, verbose active assistance cannot be used in general since the frequent highlighting is disruptive during regular

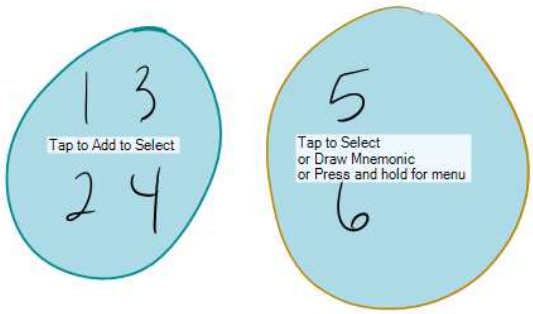

Figure 11: A lasso is drawn around the '5' and '6' when an active selection already exists for '1', '2', '3', and '4'. Both regions are shaded in blue and text prompts indicate the available gestures.

inking. We have not fully explored the gamut of design options for this class of feedback.

4.5.2 Subdued feedback. When verbose feedback is disabled, subdued feedback is generally retained. The purpose of subdued feedback is to confirm by changing the color of ink strokes the types of gesture tokens that have been recognized. This feedback, for example, is useful for indicating whether a Flick was done fast enough or whether a scribble will delete anything. Stroke coloration persists only as long as gesture tokens might possibly use the stroke to complete a gesture; as soon as the stroke cannot be part of a gesture its original color is restored.

5 FLUID BUTTONS

FluidButtons are most broadly defined as pushbuttons that, in addition to being selectable and deletable like ink strokes, support additional gestural behavior. In Section 3.2, we described how the gestural behavior of a FlowMenu can be mapped to a FluidButton. Other behaviors that we use include dragging-off, essentially drag-and-drop of a button's contents (e.g., paste button), and more interestingly, clicks and slide-offs. See Figure 12.

5.1 Clicks

Just like a conventional pushbutton, FluidButtons respond to pressing and releasing the stylus over them. However, unlike most traditional buttons, FluidButtons attempt to infer a context when none is supplied. In a traditional environment, for example, the cut button would be disabled if there were no selection, thus requiring every cut activity to be preceded by a selection activity. However, with FluidButtons, the same cut function would always be enabled because of an implicit selection rule. In our cut example, if there were no active selection then clicking the cut button would consider the last stroke drawn. If that stroke were a lasso around other strokes, then those strokes would be selected first, then cut, and the lasso deleted. Otherwise, only the last stroke itself would be cut. Even though these heuristics will not always apply, they are appealing in our gestural environment where we attempt to eliminate needless steps.

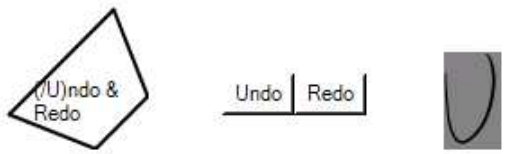

Figure 12: Three FluidButton variants for Undo/Redo created by: tearing-off a FlowMenu item; the Flick-U gesture shortcut; and tearing-off the Flick-U shortcut. In each case, clicking performs an undo, while dragging left/right performs continuous undo/redo.

5.2 Slide-offs

Unlike traditional buttons, FluidButtons support a novel interactive behavior that we call slide-offs which is loosely related to the interactive parameter setting behaviors of FlowMenus [4]. A slide-off occurs when the stylus presses down on a button and travels more than 10 pixels horizontally from the contact point. Vertical displacements are ignored so that button presses can be cancelled by releasing above or below the button as with conventional pushbuttons. Once a horizontal slide-off occurs, the button takes on a specialized interactive behavior, which can be an arbitrary function. However we have found two different classes of slide-off behaviors particularly useful: drag selection, and parameter refinement.

5.2.1 Drag Selection Drag selection behaviors are used for buttons that have an essentially unparameterized action, such as cut, delete, etc. In these cases, sliding-off triggers a selection mechanism that backtracks through time adding previous strokes to the selection set as the stylus continues to slide. In our cut example, sliding-off will initially select the last stroke drawn and then the stroke before it, etc; when the stylus lifts all selected strokes are cut.

5.2.2 Parameter Refinement Parameter refinement behaviors are used for buttons that specify scalar parameters. Clicking on the button typically sets the default value for the parameter, generally the last value set. Sliding-off the button creates a virtual slider for adjusting the scalar value. In our prototype, we use a parameter refinement FluidButton for Undo/Redo; sliding to the left performs an undo action every 10 pixels, and sliding to the right triggers redo actions. We use a similar button for setting a specific color, like red, such that sliding off the button adds or subtracts red from the selected strokes.

We have not yet incorporated parameter refinement and drag selection into the same button, although this may be possible if drag selection is assigned to horizontal slide-offs and parameter refinement to vertical slides.

6 DISCUSSION

The focus of this paper has been on techniques that can be performed with the most general hardware, a button-free stylus. This focus was motivated largely by earlier reactions to a series of designs we prototyped for specialized hardware. In particular, we watched people using gestural interfaces while eating or holding a cell phone—both impractical with any of our bimanual techniques. We also observed many users, including gesture experts, struggle with stylus-button tech-

niques especially if they had to adjust their natural pen grip. On the other hand, we noticed many users fare well with specialized hardware. Our informal observations revealed one salient trend—that the distribution of individual preference seemed somewhat balanced. Some people like buttons for good reasons, and others do not for just as good reasons. Consequently, we believe that exclusive button-free or hardware assisted techniques should always be available. In addition, optional hybrid techniques should be considered as a third option. In our prototype, the Control key instantly raises a FlowMenu when the stylus touches the display, the stylus button triggers an interactive pan and zoom mode, and the eraser can be used for erasing. Of these, only panning is frequently used, presumably because it is not yet available any other way. Among the possible additional uses for special hardware, we believe the most relevant is to dedicate a button to either disable or activate gesture recognition. This would target situations where gesture and ink can be hard to disambiguate, for example drawing a Fluid Inking gesture set diagram, or images with many dots.

During development, we performed a range of informal usability evaluations intended to identify performability concerns. We encountered three significant problems related to FlowMenus, pauses, and abstract gestures. In initial testing, we found that users did not find FlowMenus to be self-disclosing. We therefore added interactive highlighting to FlowMenus but have yet to quantify the value of this enhancement. We also observed in an earlier prototype that people generally found pausing acceptable for raising the FlowMenu, but disruptive for initiating direct manipulation. We tried a few strategies before deciding to emphasize techniques that activate widgets in place of pausing. Even though these widget gestures are not necessarily faster than pause gestures, they still seem to be preferred. We have, however, observed that some novice users occasionally trigger the resize widget when trying to perform a gesture on an active selection. This problem likely dissipates over time as strategies are developed for initially placing or avoiding the widget, although a more detailed evaluation is warranted. We also noted a striking difficulty, in an earlier prototype, that users encountered when performing and recalling an abstract gesture like the sideways, backward U that was mapped to the Undo function. When we switched to using Flick-U to activate a FluidButton, we noticed an almost complete elimination of errors.

We implemented a simple Table PC note-taking application to test the extensibility of Fluid Inking. Though far from comprehensive, this application has about 16 nameable function groups including: cut, copy, paste, delete, move, resize, save, open, lookup, infinite undo/redo, erase by stroke, zoom, duplicate, set color, and tear-off. Interestingly, when all variants are counted, the number of gestures in the interface is well more than double that of the functions, not counting the facility to create tear-off buttons for all operations. Even though we have not exhausted the possibilities, we make no pretense that Fluid Inking's simple shortcuts are arbitrarily scalable, nor do we imagine that each user will even exercise all of the shortcuts we have currently developed. Rather, we expect a user would command only a useful subset of these gestures, and this subset would vary by person analogous to how usage of modifier-key shortcuts varies by individual. However, if allowed to assign their own mnemonic shortcuts (as in many desktop applications), users may be able to optimize their gestural interaction.

Our prototype application places a strong emphasis on the regularity and discoverability of gesture shortcuts in response to the difficulty users had learning the nuances of an earlier prototype when given only a cheat sheet. We believe that just by embedding in FlowMenus a regular, albeit crude, shortcut notation (the symbols /, ○, and _, respectively denote Flicks, terminal punctuation, and pauses), we have largely addressed the problems earlier users had forgetting or jumbling gesture sequences. Similarly, we suspect that our verbose dynamic feedback, even in its early stage of development, helps to illustrate and reinforce the gesture set. We also note that the alternate subdued feedback was originally intended only for novices because we felt trained users would find it distracting; however, this feedback is left on because it provides useful feedback and is apparently less disruptive than we anticipated.

7 FUTURE WORK

Although our internal usage strongly indicates the value of Fluid Inking for some people, deeper evaluations are needed to judge its broader impact. Despite our observation of near unanimous preference for punctuated lasso select and delete gestures over their FlowMenu and hardware-button counterparts, we suspect there may be wide individual variation, akin to usage and benefit patterns found with modifier-key shortcuts for traditional menus. Since gesture awareness is a prerequisite for gesture adoption, it also is important to judge the effectiveness of our current discovery mechanisms.

Another broad area of future work is recognition of gesture tokens. For many people, Microsoft's writer-independent character recognizer (which we use for mnemonics) works quite well, but a trainable, perhaps adaptive, recognizer might support a broader range of handwriting styles and mnemonics. We have also noticed failed tap recognition was about commensurate with accidental stylus button triggering. A targeted mechanism could support recovery from failed taps, or a more sophisticated analysis of tap geometry and timing might yield more robust recognition.

We would also like to more fully examine the issues involved in mapping functionality onto shortcuts. For instance, we explored one hypothesis—that taps are best suited for non-interactive actions and pauses for interactive manipulations—however, many other other mappings are possible, such as using taps to initiate an interactive mode, or using flicks and taps to bracket compound functions.

8 CONCLUSION

We have presented Fluid Inking, an inclusive approach to modelessly integrating multi-stroke gestures with free-form inking. Our terminal punctuation and Mnemonic Flick shortcuts for FlowMenus are designed to apply broadly across hardware platforms that include small display stylus-button-free PDAs, tablet computers, and electronic whiteboards. Based on user feedback, we regularized our gesture set and

now provide both dynamic and passive feedback mechanisms to facilitate gesture discovery and to improve gesture memorability. In addition, we believe that our gesture set may be preferred over others because it uses simple, familiar mnemonics and allows for multiple interaction styles ranging from pure gesturing, to marking menus, to tearing off enhanced FluidButtons.

9 ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation, Microsoft, and the Joint Advanced Distributed Learning Co-Laboratory. Thanks to Ken Hinckley for providing helpful comments and suggestions on a draft of this paper.

REFERENCES

1. Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural computer-based sketching environment. In *Proceedings of IJCAI*, pages 1365–1371, 2001.

2. George Apitz and François Guimbretière. CrossY: A crossing-based drawing application. In *Symposium on User Interface Software and Technology*, pages 3–12. ACM, 2004.

3. William Buxton. Chunking and phrasing and the design of human-computer dialogues. In *Proceedings of the IFIP World Computer Congress*, pages 475–480, Dublin, Ireland, 1986.

4. François Guimbretière and Terry Winograd. FlowMenu: combining command, text, and data entry. In *Symposium on User Interface Software and Technology*, pages 213–216. ACM, 2000.

5. Ken Hinckley, Patrick Baudisch, Gonzalo Ramos, and Francois Guimbretiere. Design and analysis of delimiters for selection-action pen gesture phrases in *Scriboli*. In *Conference on Human Factors in Computing Systems*, 2005.

6. Takeo Igarashi and John F. Hughes. A suggestive interface for 3D drawing. In *Symposium on User Interface Software and Technology*, pages 137–181. ACM, 2001.

7. Gordon Kurtenbach and William Buxton. Issues in combining marking and direct manipulation techniques. In *Symposium on User Interface Software and Technology*, pages 137–144. ACM, 1991.

8. Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *Conference on Human Factors in Computing Systems*, pages 258–264. ACM SIGCHI, 1994.

9. James A. Landay. SILK: sketching interfaces like krazy. In *Conference on Human Factors in Computing Systems*, pages 398–399. ACM, 1996.

10. Joseph J. LaViola, Jr. and Robert C. Zeleznik. MathPad^2: a system for the creation and exploration of mathematical sketches. In *Proceedings of the 2004 SIGGRAPH Conference*, pages 432–440. ACM SIGGRAPH, 2004.

11. Yang Li, Ken Hinckley, Zhiwei Guan, and James A. Landay. Experimental analysis of mode switching techniques in pen-based user interfaces. In *Conference on Human Factors in Computing Systems*. ACM, 2005.

12. Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Symposium on User Interface Software and Technology*, pages 11–20. ACM, 2000.

13. Thomas P. Moran, Patrick Chiu, and William van Melle. Pen-based interaction techniques for organizing material on an electronic whiteboard. In *Symposium on User Interface Software and Technology*, pages 45–54. ACM, 1997.

14. Michael Moyle and Andy Cockburn. The design and evaluation of a flick gesture for 'back' and 'forward' in web browsers. In *The Fourth Australian User Interface Conference*, pages 39–46, 2003.

15. Elizabeth D. Mynatt, Takeo Igarashi, W. Keith Edwards, and Anthony LaMarca. Flatland: New dimensions in office computing. In *Conference on Human Factors in Computing Systems*, pages 346–353. ACM, 1999.

16. E. Pedersen, K. Mccall, T. Moran, and F. Halasz. Tivoli: An electronic whiteboard for informal workgroup meetings. In *Conference on Human Factors in Computing Systems*, pages 391–398. ACM, 1993.

17. Eric Saund and Edward Lank. Stylus input and editing without prior selection of mode. In *Symposium on User Interface Software and Technology*, pages 213–216. ACM, 2003.

18. Robert Zeleznik, Timothy Miller, Loring Holden, and Joseph J. LaViola, Jr. Fluid inking: Using punctuation to allow modless combination of marking and gesturing. Technical Report CS-04-11, Brown University Computer Science Department, box 1910, Brown University, Providence, RI 02912, USA, July 2004. Available at `ftp://ftp.cs.brown.edu/pub/techreports/04/cs04-11.ps.Z`.

19. Shengdong Zhao and Ravin Balakrishnan. Simple vs. compound mark hierarchical marking menus. In *Symposium on User Interface Software and Technology*, pages 33–42. ACM, 2004.