Fluid Inking: Augmenting the Medium of Free-Form Inking with Gestures

Robert Zeleznik Timothy Miller

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

ABSTRACT

We present Fluid Inking, a generally applicable approach to augmenting the fluid medium of free-form inking with gestural commands. Our approach is characterized by four design criteria, including: 1) pen-based hardware impartiality: all interactions can be performed with a button-free stylus, the minimal input hardware requirement for inking, and the least common denominator device for pen-based systems ranging from PDAs to whiteboards; 2) performability: gestures use short sequences of simple and familiar inking interactions that require minimal targeting; 3) extensibility: gestures are a regular pattern of optional shortcuts for commands in an arbitrarily scalable menu system; and 4) discoverability: gesture shortcuts (analogous to modifier keys) are displayed in the interactive menu and are suggested with dynamic feedback during inking. This paper presents the Fluid Inking techniques in the unified context of a prototype notetaking application and emphasizes how postfix *terminal punctuation* and prefix *flicks* can disambiguate gestures from regular inking. We also discuss how user feedback influenced the Fluid Inking design.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Input devices and strategies; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Interaction styles;

Keywords: gestures, button-free, terminal punctuation, inking, tablet computing

1 INTRODUCTION

Our motivation to design Fluid Inking developed from our sensing an unnecessarily large gap between interacting with the fluid medium of paper-and-pencil and with the user interfaces of current general-purpose inking applications, including those that incorporate gestures. Although we could not completely identify the factors that contribute to this distance, we intuitively felt that expansion of the free-form inking input domain to include special-purpose hardware buttons played an important role. Thus, a pervasive theme of our research was to make all functionality available through interactions that technically already exist in free-form inking. The challenge then was to implicitly distinguish interactions intended to be commands from those intended to be free-form ink. By definition this problem is not computable without knowing the mental state of the user; however, by making assumptions about the likelihood of certain interaction sequences, we believed that it was possible to make a system that was, in practice, effective.

Figure 1: 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).

To guide our design, we developed four criteria that we felt were minimally necessary to create a generally applicable approach to augmenting free-form inking with gestural interaction. These criteria include: pen-based hardware impartiality, 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 is a hybrid.

Despite the success some people have using special hardware buttons to indicate gestural commands, we were motivated to develop gestural techniques that did not *require* anything other than what inking requires, a button-free stylus. In addition to not restricting any pen-based device platforms (PDAs, tablets, whiteboards), this button-free approach also supports individuals who simply do not like using stylus or external buttons due to finger strain, awkwardness, or disruption. However, in the spirit of generality, Fluid Inking techniques can also be invoked with buttons, either as a matter of preference, or to avoid any potential recognition errors or inking conflicts.

To simplify recall and performance of a potentially (but not necessarily) large gesture set, we use multi-stroke gestures comprised of regular sequences of simple, familiar strokes. Multi-stroke sequences avoid the performance problems that arise when chunking entire actions into single physical phrases [22]. Thus, our design uses two simple patterns for disambiguating gestures from regular inking: the use of prefix *flicks* [17] (fast straight lines), or postfix *terminal punctuation* (fast taps, or short pauses). In addition, we developed techniques that extend the functionality of buttons and widgets with very simple linear gestures.

Given that conventional keyboard shortcuts are not uniformly chosen over menu and toolbar interactions, our analogous gesture shortcuts must also be considered an optional, albeit accelerated, secondary interface to functionality. We use a menu system as an extensible primary interface where gestures are visually denoted in order to be discoverable in the same way as conventional modifier-key shortcuts (i.e., there is no upfront requirement to learn the full gesture set); however, we also provide visual feedback of potential

gestures while inking both to make novices aware of the existence of gestures, and to provide feedback during multi-stroke gesture sequences.

The body of this paper details Fluid Inking's contributions in three areas, including:

- *Pen-based menu interaction*: accelerated button-free invocation; gestural shortcuts; and tear-offs.
- *Gesturing while inking*: mnemonic flicks, terminal punctuation, and speculative feedback.
- *Gestures and widgets*: abbreviated gestures; direct manipulation; widgets from gestures; pushbutton gestures.

2 Related Work

The MathPad^2 system [13] demonstrated the rudimentary notion of tap-punctuated gestures. This work broadly extends the concept through gesture regularization, pause punctuation, dynamic feedback and menu discovery.

FlowMenus [6], marking menus [10], and Scriboli [7] 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.

Moran, et al. [16] 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 predefining 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 [19]. 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.

A number of systems have explored the notion of modeless gestural user interfaces for interpreting objects in the middle of free form input. These interfaces do not require punctuation because their domain-specific drawing restrictions allow gestures to co-exist with inking. For example, Alvarado, et al [1] presented a system for creating and understanding mechanical sketches, and Gross, et al [5], described contextual recognition of gesture symbols from within free-hand drawings.

The Fly Pentop Computer [14], released as a product in the Fall of 2005, was developed in parallel with our work, and despite its lack of an active display screen, incorporates the notion of a FlyCon — a gesture similar to our technique for creating widgets from gestures. Both techniques use a mnemonic and an encircling lasso; however, our designs differ with regard to punctuation — the FlyCon requires either a double-tap or a pause, whereas ours requires an additional mimetic or mnemonic gesture symbol for the button action, followed by a single tap. The difference in our designs can be attributed largely to our increased gesture set scope — that is, we require the additional gesture symbol because we want to support more than a single action on an encircled mnemonic. Similarly, we avoid the use of double taps in a general gesture context, because their potential presence would necessarily introduce a delay in recognizing single taps. We also found that single taps were, in general, sufficient for robust recognition. Other than the similarity of the FlyCon to our technique for making widgets from gestures, our approaches are quite different.

Many other systems combine gestural interpretation with free-form annotation, such as Landay's SILK [12] and Mynatt's Flatland [18]; however, they typically use explicit prior actions (e.g., button press) to disambiguate gestures from notes. An important exception is the class of systems that use multi-modal input to distinguish gestures from each other and potentially from free-form ink. In particular, speech and gesture hybrids, such as the seminal work done in QuickSet [4], enable fluid transitions from inking to command performance, but require both suitable hardware and a suitable environment for speech recognition.

Saund and Lank [20], 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 [8]. However, their technique only avoids prior mode selection for the sub-task of forming a selection. Also, the "select" button may interfere with subsequent drawing and can require fine-grained targeting.

Mankoff, et al. [15] 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 notion that single word or phrase concepts should map one-to-one to single stroke gestures can result in gestures that are physically awkward or abstract. This unistroke phrasing also does not allow for general use of mnemonics since many symbols (e.g., "x" or "t") naturally consist of multiple strokes. Finally, Zhao and Balakrishnan [22] found relative advantages for multi-stroke gestures when comparing multiple lines to a single zig-zag for marking menus.

The GEdit system [9] did not explore the integration of free-form inking with gestures, but did present a gesture set with several parallels to our work. Its gesture set uses a marking menu, which by definition provides gestural shortcuts for accessing menu functionality, although none of the system's non-marking menu gestures were presented as menu shortcuts. GEdit also used stylus pause inflections to distinguish between interactive and non-interactive copy and drag operations, but did not use it in our sense of punctuation to distinguish ink from gesture, nor did it explore pressure-based and other timing optimizations. Although GEdit focused on unistroke gestures, it did have a single multi-stroke exception in the specific instance of distinguishing a group selection from its action, and one example of using a letter gesture to indicate an action mnemonically. Our work expands upon both of these concepts.

3 Overview

Given the scope of Fluid Inking it is useful to conceptualize the system in terms of analogies to conventional WIMP interfaces. Thus we will describe what we believe are three familiar components: a menu system, shortcuts, and widgets. First, the menu system is intended to be the starting point for users of the system. That is, a novice would use the menu to discover all functionality — in our case, we chose to use a FlowMenu [6] but believe that none of our design decisions would exclude a conventional menu or tool-

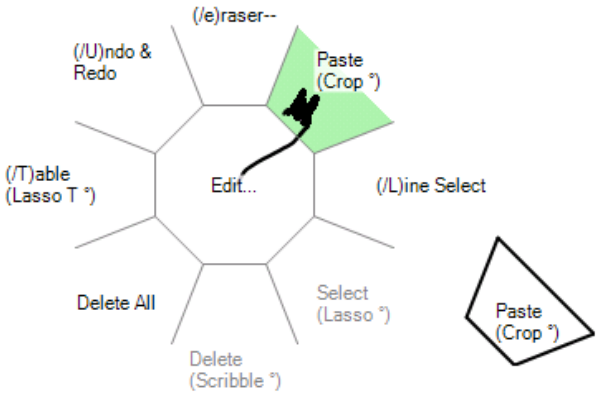

Figure 2: Menu items indicate gestural shortcuts inside ()'s. / stands for a flick, ° stands for tap or pause, and _ for pause (not seen). A torn-off Paste menu item is also shown.

bar design. Second, embedded within this menu system are visual indications of gestural shortcuts that are analogs to conventional modifier-key shortcuts. Similar to control- and alt-key variations, our gesture shortcuts come in two varieties: prefix flicks and postfix punctuation. Third, we provide a set of widgets, closely related to conventional widgets, but enhanced to support gestural interactions. That is, in addition to responding to clicks, these buttons also interpret simple gestures such as sliding off an Undo button to indicate how many actions to undo or redo.

4 Pen-based Menu Interaction

To introduce new users to Fluid Inking functionality, we need to instruct them about the existence and invocation of the main FlowMenu. If we had chosen a more conventional menu, this would not be necessary, but we felt a local menuing system was particularly appropriate for pen-based interaction. In theory, since all gestures are conceptually shortcuts for menu actions, they can be discovered through the menu system. However, in our current implementation, the cryptic indications of gesture shortcuts that we provide within the menu require additional explication. We believe that a more sophisticated display, such as Kurtenbach's [11] graphical animation of gestures could make the system fully standalone.

4.1 Accelerated Button-free Invocation

FlowMenus [6] were originally designed to be invoked with a hardware button press. Thus we adapted them by borrowing from common PDA-based press-and-hold interactions. In our implementation, we display a FlowMenu if the user presses the stylus on the display surface and holds still (stays within a 5x5 pixel box). The maximum duration of this pause is 500ms, consistent with the minimal acceptable timings found by Hinckley, et al. [7]. However, we also found that shorter timeouts were possible as a function of the pressure applied. Thus if the pressure exceeds a threshold (180 out of 255), the timeout reduces to 250ms — which, in informal testing, did not cause any additional false activations, but made menu invocation more responsive.

4.2 Gestural Shortcuts

Gestural shortcuts provide direct access to menu items and submenus. We display gesture shortcuts similarly to the way modifier-key shortcuts are shown in traditional menus (Figure 2). However, instead of "ctrl-" and "alt-" abbreviations, we use / and ° to indicate

Figure 3: 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.

flicks (Figure 3) or *terminal punctuation* (Figure 1). 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 widget (e.g., flick-U brings up an Undo/Redo widget), and we use terminal punctuation, preceded by a multi- or single-stroke drawing, for actions that directly operate on a spatial selection (Figure 1). These shortcuts are discussed in greater detail in Section 5.

4.3 Tear-off Shortcuts

We allow menu items to be torn off (i.e., copied) as persistent buttons in order to provide immediate access, to facilitate repetitive actions, to avoid hand occlusion and tension when inspecting a menu, and to reduce memorization requirements for shortcuts. While navigating a menu, users can tear off a menu item by "wiggling" the pen back and forth over the menu item (Figure 2).[1] When a tear-off event occurs, the menu item becomes a button and is dragged with continued stylus movement.

5 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. These multi-stroke shortcuts do not require an initial pause or button press since they be disambiguated from regular ink because of the design of their stroke sequences. 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 of punctuation, stroke ordering, and stroke dynamics (Figures 1 and 3). When conflicts occur, undo can be used to recover from an accidental recognition, and either undo or delete for failed recognition. Alternatively, if external or stylus buttons are available, users can optionally *require* that one be pressed during or to terminate the gesture sequence.

Besides being generally 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., the letter 'O' and lasso tokens.) This contrasts with unistroke gesture catalogs where every gesture stroke must be geometrically distinct from all others and perhaps from regular inking as well.

[1] A "wiggle" is detected when more than 6 cusps (sharp direction turns) occur within a 50x50-pixel bounding box in less than half a second. The cusps also must form a tight zig-zag, with each cusp more than 50% farther from its neighbors than they are from each other.

Gesture Class	Context	Gesture	Terminal	Example
Mnemonic flick	flick (/)	letter		ſ saves the file
Punctuated:				
self-contained		lasso (), scribble (), or crop ()	tap or pause	deletes ink under it
mnemonic	lasso ()	letter or scribble ()	tap or pause	copies ink contained in the lasso
mimetic	lasso ()	stroke hook ()	tap or pause	applies NE menu option to lasso contents

Table 1: Overview of gesture classification sequences.

Based on the structure of gesture sequences, we distinguish two classifications of shortcuts (Table 1): mnemonic flicks have a prefix flick, and terminal punctuation gestures have a postfix tap or pause.

5.1 Mnemonic Flicks

We adapted Moyle, et al.'s use of flicks [17] to work in a free-form inking context. Naive application of flicking did not work because of frequent ambiguities, such as trying to distinguish a flick from a quickly drawn '1'. However, flicks can be disambiguated from regular inking when used as part of a multi-stroke gesture. Mnemonic flicks are an extensible, easy-to-learn, gesture mechanism analogous to modifier-keys. A mnemonic flick consists of a line drawn quickly from bottom-left to upper-right, followed by an overlapping mnemonic character (Figure 3). The directional restriction is not strictly necessary but, in our testing, provided better disambiguation from general inking and was simpler than overloading flick direction with functional meaning. Representative examples of mnemonic flicks are flick-S to show a save dialog, flick-U to display an Undo widget, and flick-e to change the stylus mode to an interactive eraser. Note that the spatial location required by the first two functions is set to the start of the flick.

5.2 Terminal Punctuation

The structure of terminally punctuated gestures is straightforward — with one or two strokes, indicate a spatial context and a gesture function, and with terminal punctuation inside the preceding strokes[2] invoke the gesture. Despite its simplicity, this structure affords considerable design latitude.

We have explored three different forms of terminal punctuation: pausing, tapping, and clicking a special hardware button (e.g., control key). We define pause punctuation to be holding the stylus still (within 6 pixels) at the end of a gesture stroke for between 200ms and 500ms depending on stylus pressure and gesture context. We define tap punctuation as a fast tapping motion of the stylus against the display. Although we believe that individual users should train the pause and tap recognizers for optimal results, we have found that writer-independent recognition algorithms work reasonably as long as the concept is understood (i.e., tapping is a fast "tap", not a deliberate "press" and release or a "dot"). Our tap recognizer considers a number of features including among others the stroke's bounding box, arc length, and temporal duration. These features enable recognition based on correlations between tap timing and geometry; for instance, as user's move the stylus faster, they are more likely to input taps with the qualitative appearance of lines instead of points. Last, since there inevitably can be situations where tap and pause interpretation fails (including when depicting the Fluid Inking gesture set itself), we treat clicking the control key as tap punctuation at the last known stylus location, and pressing-and-holding the key as pause punctuation.

[2]For ease of learning, this rule is stated generally; however, it is only enforced for some gestures. In fact, advanced users may enable an option that further parameterizes some gesture functionality based on the tap location within or outside the previous strokes.

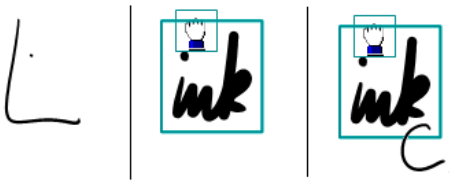

Figure 4: Left-to-right: A crop stroke is drawn indicating a paste action. The paste rectangle's lower-left corner is aligned with the crop stroke's corner. The pasted selection is copied by drawing a 'C' over it.

In addition to merely punctuating the end of a gesture, terminal punctuation can also parameterize gesture functionality. For example, we use taps and pauses to distinguish between one-shot and interactive command variants. The notion here is not that pausing, for instance, is ideally suited to invoking an interactive command variant, but rather that it simplifies the gesture set design and makes it easier for the user to recall both batch and interactive alternatives. In fact, depending on application work flow, it might be more appropriate to map tap and pause parameterizations differently, for instance if performing multiple interactive actions in a row were the expected case. Nonetheless, we believe that our mapping for tap and pause punctuation is effective when considered from the perspective of gesture recall. When performance is their criteria, we expect users would avoid the pause alternative by learning a specialized widget or gesture alternative.

Terminally punctuated gestures come in two forms: *self-contained* gestures indicate both an operand and an action in a single stroke, while compound *mnemonic* and *mimetic gestures* separate operand and action specification into two or more strokes (Table 1).

5.2.1 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[3] and then tapping or pausing within the crop's convex hull (see Figure 4). The requirement for tapping inside the crop marks is always enforced to avoid conflicts with general inking; for instance, when writing the number "7.0". Pause punctuation engages a variant of paste by presenting a menu of paste buffers.

The select gesture is indicated by drawing a lasso[4] around a group of objects. Tap punctuation is not required to be within the lasso. Pause punctuation selects the strokes and then raises an interactive FlowMenu.

[3]The crop mark may be drawn in any of the four cardinal orientations (, , ,); conceptually, the user is drawing one corner of the bounding box of the things to be pasted, and it is this corner of that bounding box which is aligned with the corner of the crop stroke.

[4]Any curve traveling at least 25% of its arc length away from its start and then returning within 25% of its arc length from its start.

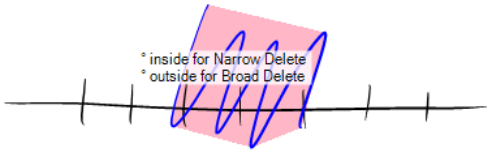

Figure 5: Tapping inside the scribble deletes the contained ticks. Tapping outside deletes the axis as well.

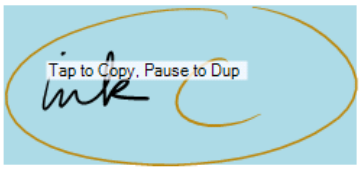

Figure 6: Mnemonic copy gesture.

The delete gesture is described to the user as first scribbling[5] back and forth over strokes. Tap punctuation is not required to be within the scribble. Pause punctuation invokes an interactive eraser mode that terminates when the stylus is lifted. Figure 5 shows an enhanced variant of this gesture.

5.2.2 Mnemonic Punctuated Gestures.

Mnemonic punctuated gestures specify actions on lasso selections mnemonically. A representative example of a mnemonic gesture is Copy. After drawing a lasso around a selection of strokes, they can be copied to the clipboard by writing an overlapping mnemonic 'C', followed by punctuation (Figure 6.) Tap punctuation copies the strokes. Pause punctuation duplicates them—that is, copies, pastes and initiates a transient interactive drag mode until the stylus lifts. 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.

It is important to note that pause punctuation variants, when considered in isolation, may not be as efficient as having dedicated gestures, such as a 'D' mnemonic for initiating an interactive duplicate without a pause. However, when viewed from the perspective of a large gesture set, we believe that the deficiencies of pause punctuation are balanced against their overall advantages in terms of simplicity and regularizing the learning and recall of command variants.

5.2.3 Mimetic Punctuated Gestures.

Mimetic punctuated gestures are actions on selections that imitate some other interaction. Our system supports mimetic interactions, reminiscent of marking menus, that mimic interacting with a FlowMenu. The benefit of the mimetic gesture is that, since the FlowMenu never actually appears, the normal timeout for raising the FlowMenu is not required. That is, the user just draws a lasso, a mimetic *stroke hook* (i.e., the same straight line doubling back on itself that would be drawn to invoke an item from a visible FlowMenu), followed by punctuation. For example, a stroke hook up

[5][21] provides a detailed description of scribble recognition

Figure 7: Cut gestures using a mnemonic 'X', and a mimetic FlowMenu interaction (Cut is in the NE octant).

to the right indicates 'cut', since 'cut' lies in the northeast FlowMenu octant. See Figure 7. We also support mimetic interactions for moving strokes.

5.3 Speculative Feedback

Although we expect that many users will discover gesture shortcuts by navigating through FlowMenus, we wanted a suggestion mechanism that could adventitiously direct novices to potential gestures. We resolved on two mechanisms, a verbose technique targeting true novices, and a subdued approach for general use.

5.3.1 Verbose feedback

As strokes are drawn on the screen, a verbose feedback mechanism highlights partially drawn gestures. For example, a lasso drawn around strokes will be shaded semi-transparently (Figure 6) to indicate tapping will complete a selection gesture. Text prompts are also displayed to describe other potential gestures starting with a lasso.

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 inking.

5.3.2 Subdued feedback.

Subdued feedback indicates the types of gesture tokens that have been recognized with stroke color changes. For example, a correctly recognized flick will turn red and a scribble that can delete something turns blue. These stroke colorations persist as long as the stroke can possibly be used to complete a gesture; as soon as the stroke cannot be part of a gesture sequence its color is restored.

6 Gesturing and Widgets

Since gesturing while inking requires prefix or postfix punctuation, widget interactions, at the cost of spatial modes, provide a useful, complementary alternative. We identified three opportunities for incorporating widget-based gestures into Fluid Inking including: abbreviated gestures, direct manipulation, and pushbutton gestures. In addition, not only do we allow menu items to be torn-off, but we also allow drawn gestures to be converted into buttons.

6.1 Abbreviated Gestures.

To simplify multiple actions on an ink stroke(s), we support abbreviated gestures on active selections. An active selection is created by lassoing a stroke(s) and tapping. This selection is a spatial context in which only gestures can be drawn and not ink. Thus, all gestures previously requiring a lasso and terminal punctuation can now be abbreviated to omit the lasso and punctuation. Pause punctuation, however, is still needed to distinguish an interactive action, although the pause timeout is shortened from 500ms to 200ms, making pause interactions more attractive since they feel more like

Figure 8: Drawing through a TranScaler widget from outside scales the selection about the corner opposite the handle. Drawing from within moves the selection.

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.

We note that, ignoring punctuation, some gestures may be subsets of other gestures. In these cases, gesture abbreviations cause conflicts which, in our current implementation, gesture set designers must avoid. We considered using timeouts to allow the longer gesture sequences to be entered, but we were not satisfied with the delay this imposes on recognition of shorter sequences.

6.2 Direct Manipulation.

To move ink, we initially provided only a tap-based non-interactive mimetic gesture and its pause-based interactive counterpart. Neither proved sufficient since mimetic movement did not afford precision and pausing was disruptive for some users. We designed the TranScaler widget to address these shortcomings and to add scaling.

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 scale, the stylus must start outside the widget and cross into it along an approximately straight line. When a crossing is detected, the selection interactively scales about the corner opposite the widget (or a previously chosen point) (Figure 8). 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 TranScaler 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 TranScaler occurs when an intended gesture accidentally starts on the TranScaler or passes through it along an initial straight line, causing inadvertent drags and scales. These conflicts can be avoided by initially tapping away from the selection or by removing the TranScaler by scribbling over it.

6.3 Widgets from Gestures

To complement the ability to tear-off a menu item, we also provide a mechanism for a gesture to be directly converted into a widget that resembles Landay's techniques [12] for sketching interfaces. We extend Landay's work by facilitating end-users in creating, using, and deleting mini-interfaces on the fly while in a free-form inking context. In addition to possibly being more direct than tearing off menu items, creating widgets from gestures naturally indicates the size, and location of the widget. Widgets can be created for any gesture by inserting, before the gesture's terminal punctuation, a lasso around the gesture and writing 'B' (or choosing "Button" from the menu.) For example, a delete button would be created by first drawing a delete gesture scribble, then lassoing it, writing 'B', and tapping.

6.4 Pushbutton Gestures.

All buttons in Fluid Inking are FluidButtons which, in addition to being selectable and deletable like ink strokes, support additional gestural behavior. For example, instead of using a pause to bring up a FlowMenu, the FlowMenu's behavior can be mapped to a FluidButton to provide instant access, or a paste button can be dragged-and-dropped. In addition, we support two classes of *slide-off* behaviors: drag selection, and parameter refinement. A slide-off behavior is triggered 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.

6.4.1 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. For example, sliding-off a "cut" button initially selects the last stroke drawn, then the stroke before it, etc; when the stylus lifts all selected strokes are cut.

6.4.2 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. For color setting buttons, sliding off the button adds or subtracts the color from the selected strokes.

7 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 techniques 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 provide a balanced gesture design that can be used without any hardware buttons at the cost of potential recognition errors, or with hardware buttons at the cost of generality and awkwardness for some users.

During development, we performed a range of informal usability evaluations intended to identify performability concerns. Our initial concerns with the concept of terminal punctuation, in terms of performance, ambiguities with free-form inking, and recall were not substantiated. Although novice users would frequently forget to tap to terminate a gesture, this problem disappeared rather quickly as it became a habit. In fact, people reported that they would perform, for example, the scribble and tap delete gesture in other applications, not realizing that those other applications did not incorporate our Fluid Inking gesture set. We also noted that, after only a brief initial training period, users did not appear to encounter any significant tap recognition problems[6]. However, we did encounter two significant issues related to pauses and abstract gestures. In an early prototype, we observed that, although pausing to raise a FlowMenu was well accepted, some users found it disruptive for initiating direct manipulation. Our solution, to provide techniques that activate widgets in place of pausing, appears not necessarily to offer a performance gain, but does seem to be more intuitive and more generally 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 early prototype, that users encountered when performing and recalling the abstract gesture of a 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 Tablet 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. Specifically, early users had difficulties caused by forgetting or jumbling some gesture sequences. In response, we embedded a regular, albeit crude, shortcut notation (the symbols /, °, and _, respectively denote flicks, terminal punctuation, and pauses) into FlowMenus that we believe should largely address their problems. 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.

[6]A very few individual users did have occasional problems getting the system to recognize their taps, but we expect making the tap definition trained per-user should greatly reduce this issue.

8 Conclusion

The Fluid Inking approach augments the medium of free-form inking with gesture commands by recognizing punctuated sequences of largely familiar inking interactions. By having the same hardware requirements as free-form inking, Fluid Inking techniques are applicable to all pen-based devices ranging from PDAs, to Tablet PCs, to electronic whiteboards. Additionally, to make a broad scope of gesture functionality easier to conceptualize, Fluid Inking techniques were designed to be analogous to conventional GUI interactions. In essence, Fluid Inking's prefix flick and postfix punctuation enable fluid intermingling of free-form inking with pen-based alternatives and enhancements to the concepts of menus, modifier key shortcuts, and interactive widgets. 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.

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 discussions.

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] P. R. Cohen, M. Johnston, D. R. McGee, S. L. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. QuickSet: Multimodal interaction for distributed applications. In *Proceedings of the Fifth International Multimedia Conference (Multimedia '97)*, pages 31–40. ACM, 1997.

[5] Mark D. Gross and Ellen Yi-Luen Do. Drawing on the back of an envelope: a framework for interacting with application programs by freehand drawing. *Computers & Graphics*, 24(6):835–849, December 2000.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Gordon Kurtenbach, Thomas P. Moran, and William Buxton. Contextual animation of gestural commands. In Wayne A. Davis and Barry Joe, editors, *Graphics Interface '94*, pages 83–90, 1996.

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

[13] 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.

[14] LeapFrog Enterprises, Inc. The Fly Pentop Computer User Guide. Part of the Fly Pentop Computer, 2005.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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`.

[22] 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.