# Stylus Input and Editing Without Prior Selection of Mode

**Eric Saund**
Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto, CA, 94304
saund@parc.com

**Edward Lank**
Computer Science Department
San Francisco State University
1600 Holloway Ave., San Francisco, CA, 94132
lank@cs.sfsu.edu

**ABSTRACT**
This paper offers a solution to the *mode* problem in computer sketch/notetaking programs. Conventionally, the user must specify the intended "draw" or "command" mode prior to performing a stroke. This necessity has proven to be a barrier to the usability of pen/stylus systems. We offer a novel *Inferred-Mode* interaction protocol that avoids the mode hassles of conventional sketch systems. The system infers the user's intent, if possible, from the properties of the pen trajectory and the context of the trajectory. If the intent is ambiguous, the user is offered a choice mediator in the form of a pop-up button. To maximize the fluidity of drawing, the user is entitled to ignore the mediator and continue drawing. We present decision logic for the inferred mode protocol, and discuss subtleties learned in the course of its development. We also present results of initial user trials validating the usability of this interaction design.

**Keywords**
Pen, stylus, sketch, draw, command, mode, InkScribe, Inferred-Mode protocol.

**INTRODUCTION**
Modern drawing programs are of two types, structured graphics programs, and sketch programs. *Structured graphics* programs, such as Microsoft PowerPoint, Corel Draw, and Adobe Illustrator, are primarily designed for interaction with a mouse. *Sketch* programs are primarily designed for stylus interaction; their goal is to simulate the freeform writing of textual notes and sketching of drawings afforded by a physical pen or pencil on a surface. Examples include paint programs, the Tivoli electronic whiteboard program [7] and the notetaking applications for the Apple Newton and the Microsoft Tablet PC. The primary data type consists of *digital ink*, where the basic unit is the *stroke*, which records the spatio-temporal trajectory of the pen/stylus (and sometimes additional properties) from touchdown to lifting from the surface.

A classic problem faced by all sketch programs is the *mode problem* [13]. In order to take advantage of the computer's ability to modify existing strokes using spatial gestures, the pen's function must be overloaded to serve multiple purposes. In draw mode, pen strokes become literal markings of digital ink. In command mode, pen strokes represent abstract gestures, which in turn can carry a number of alternative interpretations, the most common of which is the selection gesture. The mode problem results from the fact that the user must perform a deliberate prior mode-switching action. This action could be to press a button on the stylus, to double tap the stylus, to tap a toolbar button, or to touch the stylus to the screen and hold it motionless for a period of time until the cursor indicates that command mode has been entered. The problem occurs when users neglect to set the intended mode prior to performing their draw or command stroke: they find themselves drawing spurious strokes or meaningless command gestures, depending on the direction of the error. To recover, the user must disrupt their task and devote attention to repairing their digital ink content and/or getting into the intended mode.

Our prototype solution for this problem focuses on permitting the user to either *draw* or *select* image material without prior mode specification, on the assumption that selection is the primary avenue to other operations such as moving, rotating and scaling, copying, etc. Extension to other immediate command operations such as scratch-out erasing is possible but not currently implemented in our system.

**RELATED WORK**

One approach to the mode problem is to provide clear depiction of mode to the user. A recent example is the altering of prompts in PDA devices to indicate upper case vs. lower case mode [10]. The presence of a mode indicator in a visual or other modality makes little difference, however; when users are attempting to select an object, they focus their attention on that object, i.e. on what the desired effect is, rather than on the indirect matter of the state of the program that may be required to achieve that effect [12].

Some investigators have explored quasi-modes, or temporary modes invoked by explicit motor actions [9]. For example, Mohammed and Fells used a foot pedal in an interface to control music sequencing software [5].

Flatland [6], and PostBrainstorm [2] require users to press a pen button to enter command mode. This interaction style suffers from at least two drawbacks. First, on some devices, for example large electronic whiteboards and modern Tablet PCs, physically managing to press or not press a pen button while writing and gesturing can be awkward. Second, quasi-modes still require the user to perform a deliberate mode-setting action in addition to the motion that reflects the actual drawing stroke or selection gesture they intend. The cognitive load of remembering to perform this prior step purely for the sake of the interface is not relieved.

Another approach is to employ different physical hardware for draw and edit functions. SmartBoards, Mimios, and other devices provide physical eraser objects distinct from electronically sensed pens, and some Tablet PC pens have an active eraser tip as well as a drawing tip. Cosmetically, these can potentially simulate the user experience exhibited by physical counterparts to the electronic devices. We cannot pass final judgment on the usability of these approaches, but we do note that artificial constraints – such as having to replace a pen to the proper color tray, or being required to erase physical ink *only* with the electronic eraser – do create some usability problems. Alternative interaction techniques bear investigation.

A fourth approach to the mode problem is the use of specialized gestures to perform direct manipulation. One system uses a shorthand notation to create content and then allows users to edit the content [2].

Some systems such as the Apple Newton support an immediate delete command indicated by a scratch-out gesture. To perform normal selection with a Newton the user must still deliberately switch modes by holding the pen motionless for an extended period of time to get into command mode. This approach has been widely adopted in modern Tablet PC software. Once in command mode, selection is done by drawing the pen (with the highlighting mark showing as the cursor) across a single object, or by encircling multiple objects. This option does not reduce the need to monitor modes. A user must still be aware of the mode of operation of the application, and mode errors remain a hazard. Holding the pen motionless has also been adopted by Pook et al. to activate a type of marking menu called control menus [8].

In our Inferred-Mode protocol, a mediator is used to resolve ambiguity. Mankoff et al provide a taxonomy of mediators [4]. Our system uses the Choice mediator. Extending the work of Mankoff et al., the inferred mode protocol shows how a recognition strategy addresses not only data entry, but also handles recognition ambiguity at the draw/command mode boundary.

## THE INFERRED-MODE PROTOCOL

The Inferred-Mode protocol allows the user to perform either draw or command gestures without having to perform a deliberate prior action specifying the intended mode. Instead, after each stroke the program attempts to determine what the intended mode was from context. When the user's gesture is ambiguous the system presents the user with a localized pop-up button that activates the selection command (and other commands under possible extensions to our system). See Figure 1.

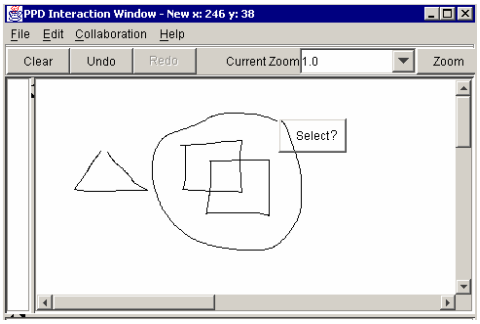


**Figure 1. Handling ambiguity in user gestures.**

Figure 2 depicts a flowchart summarizing the mode inference and mediation logic.


**Figure 2. Mode inference and mediation logic.**

One of the key differences between inferring the delete command from a scratch-out gesture (as in the Newton) and inferring object selection, is that it is critical to give users multiple command options with regard to selected material, including the ability to modify their selection, that is, to incrementally add objects to the selected set. This is important for example if the user wishes to rotate a spatially distributed set of objects about a common pivot. As is standard practice, objects that are selected are so indicated in our system by a visual highlight or halo.

Accordingly, several branches in the decision tree hinge on whether some image objects are already selected. If so,

rotate/scale handles are superimposed on the canvas along with a small button which, when tapped, brings up a menu of additional operations. The user can: 1. tap the button to bring up the pop-up menu; 2. place the stylus on a rotate/scale handle and drag to resize or rotate it; 3. place the stylus on selected material and drag to translate it; 4. start a gesture in free space (the background).

In the case that a gesture has commenced in free space, upon completion (pen-down/drag/pen-up) the system distinguishes between a tap and an extended stroke on the basis of path length. In the case of a tap, depending on location it can cause image objects to be selected or deselected. Taps on unselected objects cause them to become selected, while taps in the background cause all objects to become deselected: the ability to de-select by tapping in the background amounts to a readily available "reset" operation. We have experimented with providing the ability to selectively de-select a subset of the selected objects, but have found that while this is demonstrable under experimental test conditions, the interaction proves unreasonably complex and unpredictable in actual use.

If the user draws a long gesture, i.e. a stroke, then the system examines the characteristics of the stroke. If the stroke is an open gesture, or if the stroke is a closed loop but contains no objects, the system infers that the user's intent for that stroke is to draw digital ink.

We employ a simple heuristic shape analysis algorithm to determine whether a stroke is considered closed or not; to support fast and fluid selection it need not be perfectly closed in a topological sense. To determine whether image material lies within an encircling gesture, we calculate a tight convex hull for a closed stroke and then use a raster coloring algorithm to determine containment. We also support selection and editing of imported bitmap imagery, and some image processing is required to determine whether a closed (or nearly closed) path actually encloses a portion of a bitmap. Digital ink and bitmap objects in the vicinity of the stroke must be considered, and if any is only partially contained within the encirclement, their intersections with the encircling path must be determined and these objects split into "inside" (enclosed) and "outside" (excluded) parts. The processing requirements are nontrivial but are handled ably by modern personal computers.

If a gesture is a closed loop containing image material, the situation is possibly ambiguous as to whether the user wishes to draw a literal circle around the material, or select it through encircling. A pop-up button labeled, "Select?" appears near the end of the stroke (see Figure 1). If the user intends for the stroke to select by encircling, they tap the button. Otherwise they are free are to ignore it and keep writing or drawing, in which case the pop-up button disappears and the stroke is rendered as digital ink.

In the case that some image material is already selected, it should be possible to augment that selection by encircling additional objects. However, our experience reveals a complication. It is quite common for users to select something in one part of a scene, modify it, and then attend to some other part of the scene, neglecting to tap in the background to deselect, or reset, the selection state. They do not intend for subsequent selection and editing operations to include the already-selected and modified material, yet they commonly fail to notice that it is still selected, and they inadvertently include it in the later selection and modification operation. Therefore, our decision logic includes a branch that determines an encircling gesture to be a selection augmentation command only if it is preceded by a selection command (either a click to select or selection using the mediator). If, instead, the previous action manipulated selections, the system de-selects the previous selection and, if the stroke is closed and contains objects, displays the “Select?” button.

Because the pop-up button may be disregarded, it is minimally disruptive. The occasion in which the button appears spuriously to the user's intent occurs when the user intends to draw a closed or nearly closed path of digital ink enclosing other image material (see Figure 1). In this case, the button becomes a visual distraction. Cognitive distraction decreases with time as users learn to ignore the button when their intent is not to select. The most serious potential for disruption occurs when the button occludes a portion of the canvas where the user wishes to initiate another drawing stroke. For this reason it is important that the pop-up button appear in a location somewhat removed from the where the pen is lifted. In our early experience, spurious appearances of the mediator occurred most often during fluid writing which contains many small closed strokes. We alleviate this problem by modifying the enclosure detection criteria to disfavor very small closed paths entered in a quick succession of strokes.

Our Inferred-Mode protocol is implemented in a prototype stylus-based drawing tool called InkScribe. InkScribe also incorporates additional features to enhance the usability of pen computers, including the uniform treatment of digital ink and scanned input and the grouping of digital ink and bitmap objects using perceptual organization as discussed in [11].

## USER TRIAL

We conducted a preliminary user trial of the Inferred-Mode protocol, contrasting it with a standard version of our sketching/drawing program. The standard version used a conventional prior mode-switching paradigm in the form of buttons at the top of the screen. All other aspects of program behavior were the same.

Our user trial involved seven users performing a set of tasks during a half-hour session. Users were given an introduction to each interface during which we described the behavior of the interface and allowed the users to

practice. The order in which the interfaces were presented to the users was varied to avoid biasing the results. During the experiment, the users were given five tasks which they performed with each interface. Each task had two components and spanned two pages. The first part of the task, on the first page, asked the user to sketch some content. The second part of the task asked the user to rearrange and add to the content they initially created. A representative task was as follows:

1. Write down a series of numbers (which we provided) on the tablet computer in the order presented.
2. Rearrange the numbers in increasing order, circling the even numbers.

Users were given both a written description of the task, and a pictorial representation of what the canvas should look like after completion. We observed users and counted the number of mode errors made. We concluded the trials with a short interview where we asked the user which interface they preferred and what they liked/disliked about each.

Four of seven users preferred the Inferred-Mode protocol, even after only 1/2 hour of use.

Although difficult to draw strong conclusions from our sample set, data generated by our initial user trial is noteworthy. No user mode-setting errors were made in any Inferred-Mode trial (and indeed, the protocol makes it difficult to do so). In the mode-based interface, the number of mode errors varied from zero to five. Two users made no mode errors, one made five, one four, and two made three. The two users who made no errors exhibited a tendency to over-set the modes. Even when in the correct mode, they would re-press the "Draw" or "Edit" button to ensure the correct mode. As well, it seems that the absence of errors is not an indication of user preference. Of the two users that made no errors, one preferred the mode-based interface, the other the Inferred Mode protocol.

| # mode errors | 4 | 1 | 3 | 5 | 0 | 0 | 3 |
|---|---|---|---|---|---|---|---|
| Pref: Mode/Inferred | I | I | M | I | I | M | M |

**Table 1 Summary of data from user trial.**

In more complex tasks such as brainstorming activities it has been noted that even lightweight constraints on user interaction can be burdensome [1]. We are conducting more extensive experimental trials entailing detailed analysis of the conditions under which mode errors occur. In particular, the incorporation of a distracter task would be valuable in analyzing the prevalence of errors in simple drawing tasks, and the benefits of alleviating those errors by inferring modes.

We believe that our early testing has uncovered the major critical design factors toward obviating the mode problem in pen-based sketch systems using an Inferred-Mode approach. In our design and implementation, the Inferred-Mode protocol is conservative in its ambitions about inferring user intent. Care in design and implementation are critical to avoiding the classic failure modes of DWIM (Do What I Mean) systems that are overly confident and aggressive about guessing what a user wants. We suggest that the Inferred-Mode protocol shows good promise of falling on the safe and useful side of the line.

## REFERENCES


1. J. Arvo, "Computer Aided Serendipity", *Proceedings of Graphics Interface, 1999*, 183-192.

2. F. Guimbretiere, M. Stone, and T. Winograd, " Fluid Interaction with High-resolution Wall-size Displays," Proceedings *UIST'2001*, ACM, Orlando, November 2001.

3. G. Kurtenbach and W. Buxton, "The Limits of Expert Performance Using Hierarchic Marking Menus", *INTERCHI '93*, 482-487.

4. J. Mankoff, G. Abowd, and S. Hudson, "OOPS: A Toolkit Supporting Mediation Techniques for Resolving Ambiguity in Recognition-Based Interfaces", *Computers and Graphics*, 24:6, 2000, 819-834.

5. F. Mohamed and S. Fels, "LMNKui: Overlaying computer controls on a piano controller keyboard", *CHI 2002,* 638-639.

6. E. D. Mynatt, T. Igarashi, W.K. Edwards, and A. LaMarca. "Flatland: New Dimensions in Office Whiteboards", Proceedings of *CHI'99*, pp. 346 - 353.

7. E. Pedersen, K. McCall, T. Moran, and F. Halasz, "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings", *INTERCHI '93*. ACM, New York, 1993.

8. S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot, "Control menus: excecution and control in a single interactor", *CHI 2000 Extended Abstracts*, pp. 263 - 264.

9. J. Raskin, "Looking for a Humane Interface", *CACM*, 40:2, 1997, 98-101.

10. H. Ryu, "Will it be upper-case or will it be lower-case: can a prompt for text be a mode signal", *CHI 2002*, 836-837.

11. E. Saund, J. Mahoney, D. Fleet, D. Larner, and E. Lank, "Perceptual Organization as a Foundation for Intelligent Sketch Editing", 2002 AAAI Spring Symposium on Sketch Understanding, Stanford University, March 2002.

12. A. Sellen, G. Kurtenbach, and W. Buxton, "The Role of Visual and Kinesthetic Feedback in the Prevention of Mode Errors", *Proceedings of Human-Computer Interaction,* 1990, 667-673.

13. L. Tesler, "The Smalltalk Environment", *Byte*, August 1981, 90-147.