GestureBar: Improving the Approachability of Gesture-based Interfaces

Andrew Bragdon, Robert Zeleznik, Brian Williamson[†], Timothy Miller, Joseph J. LaViola Jr.[†]

Brown University
Department of Computer Science
115 Waterman St. 4th Floor
Providence, RI, USA
{acb, bcz, tsm}@cs.brown.edu

[†]University of Central Florida
School of EECS
4000 Central Florida Blvd.
Orlando, FL, USA
{jjl, bwilliam}@eecs.ucf.edu

ABSTRACT

GestureBar is a novel, approachable UI for learning gestural interactions that enables a walk-up-and-use experience which is in the same class as standard menu and toolbar interfaces. GestureBar leverages the familiar, clean look of a common toolbar, but in place of executing commands, richly discloses how to execute commands with gestures, through animated images, detail tips and an out-of-document practice area. GestureBar's simple design is also general enough for use with any recognition technique and for integration with standard, non-gestural UI components. We evaluate GestureBar in a formal experiment showing that users can perform complex, ecologically valid tasks in a purely gestural system without training, introduction, or prior gesture experience when using GestureBar, discovering and learning a high percentage of the gestures needed to perform the tasks optimally, and significantly outperforming a state of the art crib sheet. The relative contribution of the major design elements of GestureBar is also explored. A second experiment shows that GestureBar is preferred to a basic crib sheet and two enhanced crib sheet variations.

Author Keywords

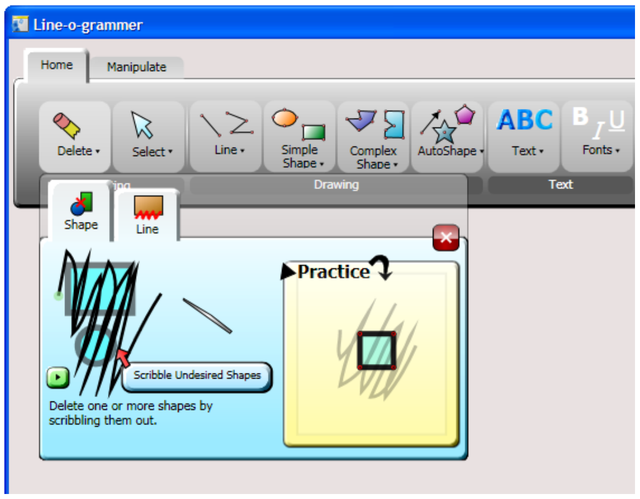
Gestures, pen, approachability, disclosure, learning

ACM Classification Keywords

H5.2 Information Interfaces and Presentation: Interaction Styles, Evaluation/Methodology

INTRODUCTION

High quality pen-based hardware devices have become increasingly available at successively lower cost. However, companion gestural UIs have gained little traction despite their strong value proposition: gestural commands physically chunk a command and its operands into a single action [4], and different commands can be intermingled without an explicit mode switch, for example, to enable transparent transitions between drawing, moving and erasing [26]. Gestures can also be committed to physical muscle memory which can help users focus on their task instead of the UI. The HCI community has a long history of developing gestural UIs which demonstrate this value, going back to [5]. Why then, do most applications forgo the potential of gestures, relying instead on conventional WIMP paradigms, such as menus and toolbars? We believe the basis for an answer lies in the refrain we commonly encounter when pitching gestural applications to software industry leaders: "this is great, but how will new users learn these gestures?"



Delete one or more shapes by scribbling them out.

Figure 1. GestureBar after clicking Delete. Commands are not executed when clicked; rather the Gesture Explorer drop-down displays an illustrative animation with detail tips, a replay button, a text description and a practice area.

We interpreted this concern broadly to mean that the research challenge for making gestural UIs mainstream is *approachability*, the summative experience of a first-time novice who attempts to accomplish a non-trivial, ecologically valid task, without human assistance, training or introduction. Thus, instead of concentrating on performance and retention metrics which apply more to users who have adopted a gestural UI, our primary focus is on developing and evaluating techniques that facilitate the acceptance of a gestural UI by a *novice* – someone assumed to be unfamiliar with the fundamental gesture notion – that “drawing” something can specify operands and execute a command.

Our approach, GestureBar (see Figure 1), embeds gesture disclosure information in a familiar toolbar-based UI. Users encounter relevant gesture details only as needed, after they have formed a mental goal, searched for, and found an appropriate command – consistent with Polson, *et al.*’s CE+ model of novice behavior [22]. When a tool is clicked, GestureBar displays feedback that richly discloses information about the gesture, and provides an area in which to experiment without impact on the user’s document. Note that clicking an item does NOT execute the command, but rather discloses the gesture, and how to perform it.

In this paper, we explore the hypothesis that gestural interfaces can be approachable – supporting a walk-up-and-use experience. First we discuss the evolution of GestureBar from a set of design principles and prototypes. We then present two user studies, conducted in the context of a full gestural application, to test our hypothesis and yield insight into the relative merits of our major design elements.

RELATED WORK

Gestural UI research spans a broad set of topics, including: creating gesture sets [16] [12], disclosing gestural functions and teaching individual gestures [15] [3], recognizing individual gestures [25] [23], and correcting recognition results [17] [14]. Although each of these areas influences the usability, our GestureBar work focuses specifically on novice approachability. We note that we are unaware of any prior evaluations that study the approachability of gestural UIs in which pure novices are given a full-scale gestural application but no *a priori* training or advice on its use.

Prior gestural UIs have commonly treated gestures as an organizing principle in themselves which can compromise searchability. For example, crib sheets often display gesture sets as gesture-icon/command-name pairs in a two-column table (*e.g.*, Mouse Gestures [20]), or a 2D grid (*e.g.*, Graffiti [21]). Within this layout, gestures may be clustered alphabetically or according to the similarity of their functions [1]. Such crib sheets, although effective as reference guides, are generally complemented by additional novice training material such as videos and interactive tutorials. The Graffiti gesture set [21] was a notable exception, as it was rapidly adopted by a broad base of users who had access only to a crib sheet. We believe Graffiti’s crib sheet was successful because it leveraged intimate *a priori* familiarity; its crib sheet may likely have been perceived as an organized collective whole – the alphabet – instead of a cluttered set of gestural commands. Fluid Inking [27] treated gestures analogously to command-key shortcuts by embedding their mnemonic description in a menu system. However, because both the mnemonic depiction of the gestures was crude and also non-gestural alternatives were available, this technique was not successful with novices. Grossman’s [7] work on accelerating the learning of hotkeys included a technique in which menu items were disabled, thus requiring users to learn and use associated hotkeys to execute commands; the technique was positively received by users and resulted in improved performance. Our work draws from this result and applies this basic design notion to the disclosure of gestures, as clicking GestureBar items does not execute commands, instead disclosing the appropriate gesture.

InkSeine [10] presented a variation of the crib sheet theme in which gestures were shown *in situ* as highlighter annotations over application widgets; the annotations could be toggled on and off with a button press. This technique was well suited toward disclosing simple gestures associated with explicit UI widgets. However, with only a few gestures, the technique cluttered the workspace but did not provide support for accessing more detailed information about subtle or complex gestures or for displaying gestures that required a document context (*e.g.*, a selection lasso).

In addition to searchability problems, the iconic representations used in crib sheets are not always effective at expressing the essential characteristics of all gestures, including the context where they apply and possible gesture variations. Kurtenbach [15] explored extending crib sheets with animation to make gestures more learnable. In this system, pressing and holding within the document invoked a contextual crib sheet and pressing a crib sheet item presented a series of animations illustrating examples of the gesture within the active document. Users could then trace the gestures to develop the physical skill required to perform the gesture correctly and execute the corresponding function. However, this system did not support demonstrations of geometrically parameterized gestures, did not have a mechanism for highlighting geometric gesture nuances, and did not support browsing through functions that required an existing context unless the user had already created that context in their document. In addition, users needed *a priori* knowledge to know about the press-and-hold “gesture” to bring up the crib sheet and special, unmarked contexts like the margins of a page. Hinckley explored a related variant in which text prompts and traceable extensions were displayed when partially-completed pigtail gestures had been entered [9].

As an alternative to crib sheet organizations, marking menus [24] and zone and polygon menus [28], are organized by the hierarchical, radial nature of the gestures they support. Grossman, *et al.* extended the basic notion of marking menus to support a broader range of gestures with Hover-Widgets [8] a technique which simultaneously depicts all available non-marking hover state gestures as paths emanating from a common starting point. Bau, *et al.* created a similar technique, OctoPocus, that depicted gestures as colored trails emanating from a common starting point [3]. In either case, users learn to perform a gesture by following its trail while receiving continuous feedback about their performance – gesture trails are pruned or reinforced based on how closely the user follows that trail. With all these techniques, gesture labels are spatially arranged based on the geometry of their corresponding gesture which often results in related labels being spatially separated. Non-hierarchical displays like OctoPocus and HoverWidgets also become cluttered as the number of gestures increases. Trail-based approaches also do not adapt well to many common gesture types (*e.g.*, short taps, fast flicks, and spatially parameterized gestures like lassos or drag-and-drop) and they cannot be used with all gesture recognizers, for instance those that do not provide incremental feedback. Color-blind users also may have difficulty interpreting OctoPocus' color codings.

A different approach to disclosing and teaching gestures is to make a clean separation between disclosure and invocation, such as with step-by-step training videos or interactive tutorials. Microsoft provides an interactive tutorial for users to experiment with Flicks and receive textual disclosure about what they did right and wrong. Forsberg, *et al.* explored an online guided tutorial, activated at any time by pressing a "Demo" button, in the Tablet PC Music Composition Tool [6]. This tutorial presented the gestures in the gesture set simultaneously as faded out annotations over a partly finished musical example; users could then trace any trail to finish that part of the score. However, these approaches come at a higher cost – they require a significant, upfront time and attention commitment from the user as well as additional production time by the system developer.

Sketch-based UIs (*e.g.*, [1][13][2]) attempt to recognize hand-drawn diagrams based on their visual appearance the way a human would. In this sense, users have less of a need to explicitly learn a UI. However, our work is still applicable to these systems since most novice users will not know all of what can be sketched (*e.g.*, the symbol for divorce in a family tree diagram [2]) or how to perform an abstract gestural command such as Zoom or Copy.

GESTUREBAR DESIGN

Design Principles

We expanded Kurtenbach, *et al.*'s Learning-While-Doing-Strategy [15] for self-disclosure by identifying four design principles that together affect application approachability:

Familiarity: UIs should conform to commonly held *a priori* knowledge and expectations, rather than requiring upfront learning to get started.

Searchability: Since a user's first priority when using a new application is to find relevant commands, the UI must facilitate command browsing and identification.

Expressivity: Unlike traditional GUIs, gestural UIs must be capable of fully disclosing compound physical interactions that are recognized by complex algorithms.

Low Cost: UIs must be practical both for the system designer and for the end user and be compatible with traditional UIs; they should not limit recognition technology, require programming effort beyond what is needed for standard UIs, or fundamentally alter user workflow.

Prototype and Iterative Design

We began the design process with a simple mockup, using Windows Presentation Foundation (WPF), designed to test the idea of leveraging the familiarity of the ubiquitous toolbar paradigm for gesture disclosure (see 1st iteration in Figure 2). Gestures were displayed as static images, each labeled with the appropriate command name, in a toolbar-like layout across the top of the screen. However, on hover, the gesture icons were animated to demonstrate the dynamic nature of the gesture. Each animation was designed as a canonical example, showing appropriate context; a shape around which a selection lasso was being drawn, for instance. We tested the mockup on three users with no Tablet PC experience; they performed a series of command execution tasks (identical to those in Experiment 1, see below) in a think-aloud protocol. We found that the toolbar did not in fact look familiar to users who were accustomed to seeing icons that depict functionality – they fruitlessly looked for "standard" toolbar icons (*e.g.*, Undo) while being continuously interrupted by small, inscrutable gesture animations. In essence, the gesture animations were confusing since they confounded the natural function-browsing workflow.

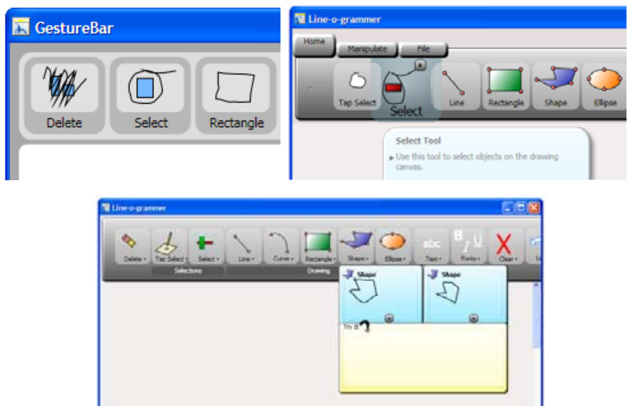


Figure 2. Iterative design process: 1st iteration of GestureBar prototype (top left), 2nd iteration (top right), 3rd iteration (bot.)

For our first real GestureBar prototype (see Figure 2, 2nd iteration), we changed the toolbar to display function icons which expand on hover to show a larger animated demonstration of the associated gesture. We performed an initial pilot test on three users who had never used Tablet PCs, with the same tasks as before, and found that although they were ultimately able to accomplish the tasks given to them, they experienced significant initial confusion. They did not expect either the on-hover behavior, or the animation; and they were puzzled why commands were not executed when they tapped on the toolbar buttons. They also missed most or all of the animations since the animations played quickly and without an initial delay. In addition, they complained that the expanded hover animation area covered nearby buttons making it hard to see the commands they needed. We adjusted the toolbar to be more familiar by getting rid of hover animations and displaying instead a Gesture Explorer dropdown when a tool item was clicked (see Figure 2, 3rd iteration). The dropdown provided more room to show a large, clear animation without disrupting the searching process or covering neighboring buttons, and is dismissed by clicking anywhere outside the dropdown or on a close button. We also added a short “attention getting” introductory animation (an expanding gray rectangle) prior to playing the gesture animation and we supported secondary animations, intended to disclose gesture variations that could be tap-activated. Another set of four pilot users in a third, identical pilot test indicated the toolbar was no longer confusing, but new problems had arisen.

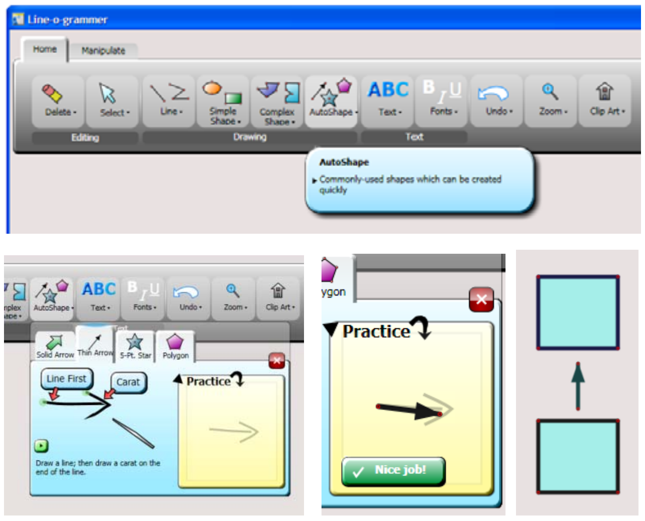
Our expectation that animations would clarify gestures was not fully supported. Users found the secondary animations to be useful, but visually overwhelming. In addition, we observed that animations did clarify directional requirements of gestures, but seemed to obscure geometric content. For example, our lasso gesture requires users to draw a loop enclosing the objects of interest that ends in a small tail, but users did not notice the tail and simply circled the objects of interest. As another example, a text gesture required an underline terminating in a sharp upward hook to be drawn below handwritten text, but many users overlooked the hook.

We concluded that no single silver-bullet strategy was expressive enough to present all nuances of even simple gestures, and that multiple strategies needed to be combined with careful attention to visual complexity. In addition, we observed that as users underwent the trial-and-error muscle training process of performing gestures, they often ended up doing significant damage to their document as failed gesture recognition led to unexpected results such as stray lines and text, and unintended command invocation.

Final Design

To support command set scalability and infrequently used commands, we added tabs across the top of the GestureBar, similar to the Ribbon in Office 2007 [18], and we added toolbar items which directly execute functions instead of displaying gestures. Switching tabs allows users to easily browse for commands in a single place (no menus are needed for less-used commands), and related sets of commands can now be grouped together. We added labeled groups within each tab (similar to the Ribbon), for example, the drawing commands might be in one group, “Drawing” and the text commands might be in another group, “Text”. In addition, we added tooltips which display on hover, showing the command name and a brief description of its use. Finally, to reinforce awareness of which gesture was recognized, we gradually fade the corresponding toolbar icon in and out of the document at the end of each completed gesture, much like Windows Vista Flicks [19] and Bau, *et al.*’s display of command associations [3].

We made several improvements to the expressivity of the Gesture Explorer (see Figure 3). Tabs are provided for switching to related gesture variations so that each tab page can show a single animation at a larger size to accommodate the addition of pen sprites which “write” each stroke, highlighting movement to the starting location of the gesture and transitions between segments of multi-stroke gestures. We added a green dot to indicate the start of each stroke and salient textual detail tips which appear via fly-in animation – so as to draw attention to them – at the end of the demonstration animation. We also added a text description to each gesture demonstration that describes the command and gives an overview of how to perform the associated gesture.



AutoShape
Commonly-used shapes which can be created quickly

Draw a line; then draw a carat on the end of the line.

Figure 3. Usage scenario: hovering over AutoShape reveals a tooltip (top), clicking AutoShape opens the Gesture Explorer; Clicking the Thin Arrow tab displays an animation with detail tips (left); a successful gesture attempt in the practice area (center); user adds an arrow to their document (right).

To facilitate muscle training without affecting the document, we added a Practice Area that shows a semi-transparent, static traceable overlay of the gesture, and application content, where appropriate – for example, lasso select provides a square for the user to practice selecting. The Practice Area is an instance of the same WPF control as the application document which makes it easy to simulate the in-document recognition experience without fear of “messing up” the document. We provide “Nice Job!” or “Not Quite Right” (see Figure 3) pass/fail notifications in response to a callback from the gesture recognizer. If an executed gesture matches a set of success gestures, “Nice Job!” is displayed. If a reported gesture is in a set of intermediate actions, no result is shown, allowing users to complete multi-stroke gestures. Finally, if a gesture is in neither set, “Not Quite Right” is displayed. Tapping this notification resets the practice area for another attempt. One can imagine proffering rich feedforward [3] and explanatory feedback about “why” a gesture failed, but this may have increased the implementation and design costs and imposed gesture class or recognition technology constraints.

Content Development

Distinct from the GestureBar design, is the development of application-specific content. Similar to WIMP menu/toolbar content, creating content for GestureBar does not involve writing code. A developer specifies content values such as icons, text, and gesture strokes as WPF properties in a GUI editor; gesture demonstration animations are procedurally generated by GestureBar using WPF. Developers must also decide which gesture nuances need reinforcement with detail tips; we were careful to provide no more than three such tips to eliminate clutter. In addition to actually specifying content, designers must also consider whether each gesture variant warrants special attention requiring its own command button (*e.g*., delete vs. clear all), and whether important gesture sequences should be treated as a single command with its own button (*e.g*., writing formatted text). This distinction is particularly relevant for gestural systems which often rely on interaction strategies that may not be obvious given only isolated gesture descriptions.

EVALUATION

To evaluate the approachability of GestureBar, we opted to study the performance of an unassisted novice user in a complex "real" application with a range of gesture types. We felt that easier-to-control synthetic tests that artificially constrained user workflow would be inadequate to assess approachability. We also implemented variant UIs to span the design space between GestureBar and a crib sheet so as to identify the relative value of our main design choices.

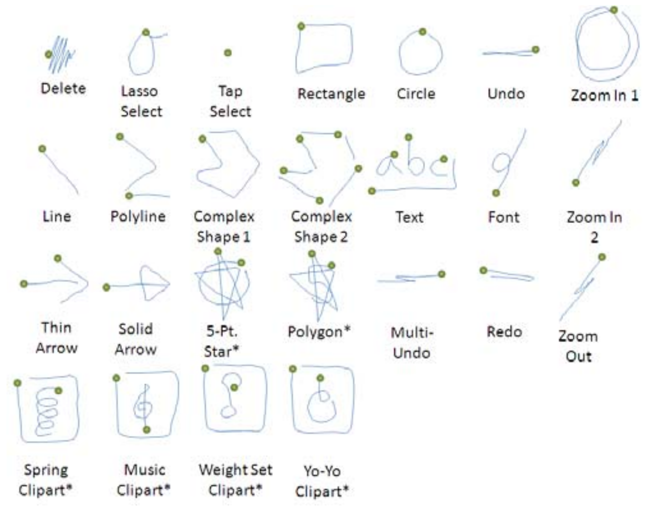


Table 1. Lineogrammer gesture set: a dot indicates where each gesture starts. Gesture variations not shown for brevity.

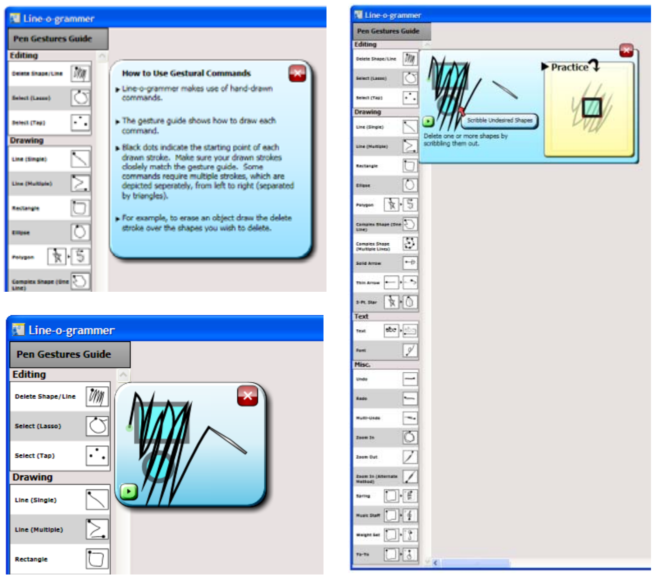
Lineogrammer Context

Our evaluations were conducted in terms of ecologically valid diagram replication tasks using Lineogrammer [26], a research system that we have developed for creating simple diagrams, that we feel is representative of a wider class of gestural applications. As expected in a gestural application, unscripted and unanticipated actions can occur when users fail to perform gestures correctly or when gestures are misrecognized. In Lineogrammer, such errors typically generate stray lines or other geometry within the user's document. Thus we were able to observe the complete open-ended process by which users approach an unknown interface, including forming goals, searching for commands, performing gestures, and assessing results.

Table 1 summarizes Lineogrammer's gestures and notes additions made for the purpose of this study to increase the generality of the gesture set without violating the essential nature of the application. For example, Clipart gestures were added to represent two important classes of gestures, multi-stroke and mnemonic; the Polygon gesture was designed as a stress test for learnability since its visual appearance and function are confounding.

Conditions

We chose to evaluate GestureBar (GBAR) relative to the status quo of a basic crib sheet (CRIB) similar to that used by the Mouse Gestures Add-on [1] to Firefox. This baseline is important because it presumably represents the problems of approachability that are perceived with gestural interfaces in general. By demonstrating a significant advantage of GBAR over crib sheets, we hope to persuade those who believe gestural UIs are inherently unapproachable that the problem may rather be a function of a particular UI style.



How to Use Gestural Commands

- Line-o-grammer makes use of hand-drawn commands.
- The gesture guide shows how to draw each command.
- Black dots indicate the starting point of each drawn stroke. Make sure your drawn strokes closely match the gesture guide. Some commands require multiple strokes, which are depicted seperately, from left to right (separated by triangles).
- For example, to erase an object draw the delete stroke over the shapes you wish to delete.

Figure 4. CRIB: crib sheet displays generic usage text on click (top left); ANIM: displays animated demonstration on click (bot. left); EXPLOR: displays gesture explorer on click (right).

During pilot testing we had to adapt our baseline from the basic crib sheet style used by Mouse Gestures to a crib sheet that displayed a tooltip (see CRIB in Figure 4) explaining the concept of a gesture and a crib sheet whenever the crib sheet was clicked on. This change was prompted by four pilot users of the basic crib sheet who were essentially unable to find or perform any gestures other than the implicit line, polyline and rectangle gestures; instead they treated the crib sheet as a toolbar and repeatedly clicked on it to change modes only to find that nothing happened. We also eliminated the confound of scrolling within the basic crib sheet since it fit the height of the display of the Tablet PC. To gain perspective on the value of the major components of GBAR, we also tested two intermediate designs that blended GBAR features into the basic crib sheet.

The first of these intermediate designs, ANIM, is the basic crib sheet extended with canonical animated demonstrations of the gestures in context (identical to the animation from the GBAR design but without detail tips). These demonstrations were shown in a small window with replay and close buttons, in response to clicking on a crib sheet entry.

The second of these intermediate designs, EXPLOR, is the same as ANIM but extended to show the full gesture explorer UI – including animation demonstrations, detail tips, a text description, and practice area.

The content used in these three conditions – command names and groups, animations, Gesture Explorer design, etc. – was taken from GBAR except for the static gesture icons which used the Mouse Gestures ink style [1].

Experiments Overview

We conducted two evaluations across our four condition set. First we performed a within-participants design to qualitatively assess which conditions the users liked, and what they liked about them. Then we used a between-participants design to quantitatively measure summative usability.

EXPERIMENT 1

Participants and Equipment

We recruited 24 participants (20 male, 4 female, ages 19-24) from the general student body of the University of Central Florida. Each participant was paid $15. Participants were required to be able to operate a Tablet PC. We advertised the study widely to get a sample of participants with diverse backgrounds and levels of experience using computers; 5 participants identified themselves as "Experts" in terms of computer expertise, the average rating across the participants was just above "Intermediate," the middle of a 5-point Likert scale. 3 participants had used a Tablet PC before, and 2 participants regularly used a Nintendo DS. All trials used an HP tc4400 Windows XP Tablet PC with 1 GB of RAM, in slate configuration.

Tasks

We created four task sets, each comprised by an ordered sequence of tasks. Each task required the use of a gesture; several examples are "make a medium-sized rectangle," "zoom in on one of the circles," "make a star autoshape in an unused portion of the screen." Each task set contained five gestures the user had not yet seen in a prior task set as well as several simple gestures (such as rectangles and squares) that the user had already seen. Not all gestures were assigned to task sets; six gestures were used as distracter commands, so that there would always be commands the user was unfamiliar with and would have to search through, even if the user successfully executed every task set.

Experimental Design

We used a 4-factor within-participants design, with the factors being the 4 conditions (see above). The condition order was counterbalanced and randomized with one order per participant. Each user performed one task set per condition, with the same task set ordering for all users.

Procedure

Participants were read an introduction describing an overview of the tasks they would be doing. Participants were asked to think-aloud, describing their thoughts and intentions out loud as they performed each task. Participants were told to go at their "normal working pace." Participants were told that the diagrams they created did not have to be "100% perfect" but to "do as well as you can." Before starting the experiment, the concept of a gestural command, and how to use one, was explained to each participant. Participants were also told that there would be a help system on the side of the screen that would show them how to perform the gestures, and that they would see four different help systems. In addition, participants were told that the person running the experiment would not answer questions about the software, but could clarify task descriptions.

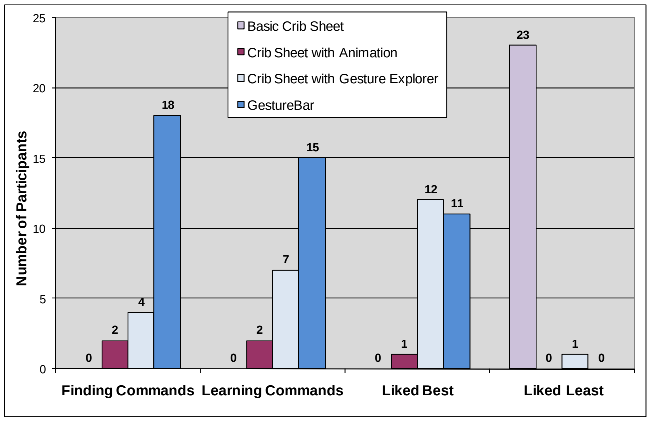
We felt that handling failed gesture attempts is an important usability aspect of any gestural system, and so coping with failed gesture attempts was left to the participants. Some participants ignored the stray lines and geometry failed attempts generated, while others sought to delete or undo it. As the purpose of the experiment was to collect qualitative feedback, we believe that this added significantly to the realism of using a variety of gesture disclosure interfaces.

After the introduction, participants were randomly assigned an ordering of the conditions. Users were then given one task set per condition. Users were asked to perform each task from each task set sequentially. If users made 10 or more failed gesture execution attempts, or if they said that they were stuck and wished to move on, they were given the option of continuing on to the next task. To ensure participants were aware of the change in conditions, before the start of tasks 2, 3, and 4, participants were told that the help system had changed and that clicking on it would produce a different result than before. A questionnaire was administered at the end of the study to assess users' overall experience using the conditions. To help participants identify the conditions in the questionnaire, a color figure was provided showing screenshots of each condition, labeled anonymously "Help System 1," "Help System 2," etc.

Results

Results from post-questionnaire data (see Figure 5) show there were significant differences in participants rating of each technique in terms of finding commands ($\chi^2_3 = 39.33$, $p < 0.05$), learning commands ($\chi^2_3 = 22.33$, $p < 0.05$) as well as which technique they preferred the most ($\chi^2_3 = 20.33$, $p < 0.05$) and the least ($\chi^2_3 = 64.33$, $p < 0.05$). Participants also highly ranked, using a 5-point Likert scale where 1 is very negative and 5 is very positive, the animations (Mean = 4.59, SD = 0.95), detailed fly-ins (Mean = 4.0, SD = 0.95) and the practice area (Mean = 4.27, SD = 0.83).

The majority of the participants felt the GBAR made it easier for them to both find and learn commands. Participants were split between GBAR and EXPLOR in terms of which technique they liked the best. Examining their responses as to why they chose a particular help system shows that the common thread among their responses was the usefulness of the animations, the practice area, and the detailed fly-ins (supported by the Likert scale questions on each component). However, the distinguishing characteristic appears to be the layout of the help system as some participants felt that GBAR was a well organized approach to providing help learning gesture while others preferred the fact that EXPLOR was laid out all once and showed gesture example icons. CRIB was almost unanimously voted as the worst help system; participants remarked that it was very difficult to find things using it and that it really did not offer any help to learning and using the gestures. Finally, overall comments regarding the four help systems indicated that several participants thought the organization found in GBAR was very helpful although they wanted the help to stay on when they needed it and to go away when they didn't. These comments suggest a hybrid approach of GBAR and EXPLOR is worthy of future investigation.



	Basic Crib Sheet	Crib Sheet with Animation	Crib Sheet with Gesture Explorer	GestureBar
Finding Commands	0	2	4	18
Learning Commands	0	2	7	15
Liked Best	0	1	12	11
Liked Least	23	0	1	0

Figure 5. Disclosure technique preferences of participants.

EXPERIMENT 2

Participants and Equipment

The recruitment procedure and equipment was the same as for Experiment 1. We recruited 44 participants (27 male, 17 female, ages 18-33) from the general student body of Brown University. In addition, to ensure that users met our definition of novice, after a participant completed the experiment, we read them a description of a pen gesture, and asked if they had ever encountered anything similar prior to the experiment; none had. 11 participants had used a Tablet PC before and 1 participant regularly used a Nintendo DS.

Tasks

We created four tasks for Experiment 2. Tasks 1 and 2 asked a participant to replicate a given diagram (see Figure 6). Tasks 3 and 4 asked participants to replicate a diagram and then to change it to look like another given diagram.

Each task was designed so that there was a single, unique, optimal combination of gestures that would produce the highest quality result with the least effort. Tasks were specifically designed to not require artistic ability. Each task, to be done optimally, required learning a specific set of new gestures (the task learning set); we distributed the gestures in Lineogrammer evenly across each task, with six distracter gestures left unused. Non-optimal methods, *i.e.* those which did not use, at a minimum, all of the gestures in the task learning set, were fundamentally worse than the optimal method; they typically required significantly more effort and produced an obviously substandard result.

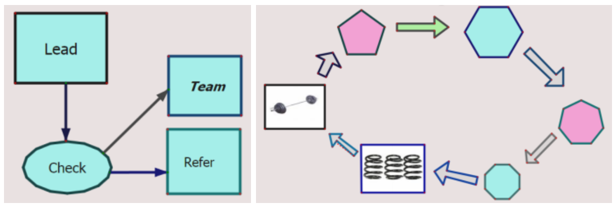


Figure 6. Task 1 (left), task 2 (right) from Experiment 2.

This approach allowed us to give users rich tradeoffs between accomplishing a task in a substandard and inefficient way, or by searching for an appropriate command to make the task easier, learning the gesture, and executing it. For example, users had to choose between making an arrow shape by free-handing it (resulting in an obviously lopsided arrow), or by searching the application for an appropriate command, finding the solid arrow command, learning the solid arrow command, and executing it, to make a perfect solid arrow, identical to the one depicted in the task. In another example, a user might have to choose between redrawing a complex item (such as the robot from task 3) from scratch after deleting it, and searching for and learning how to use an undo command to undo the deletion.

The single-part tasks gave a sense of how users responded to each condition when given no direction; the multi-part tasks created scenarios in which commands such as Delete, Undo, and Zoom In were part of the task learning set.

Classifying Gesture Attempts

We observed each participant's interactions with the software and identified each gesture attempt, as defined here:

Gesture Attempt: An instance of a user attempting to perform a gesture in the document (practice area not included).

We identified the gesture participants had attempted to perform by the command they had tapped on, their own think-aloud verbalizations, and the structure of the ink they wrote. In all cases user intent was clear from the information available, and the distinctiveness of each gesture. Gesture attempts were then classified as either failed or successful.

Metrics

Discovery Rate: The number of unique non-distracter gestures attempted (successfully or unsuccessfully) divided by the total number of non-distracter gestures.

Overall Coverage Rate: The ratio of successfully performed to total number of non-distracter gestures.

Performance Category: For each gesture in the combined task learning set, for each participant, we assigned one of five categories: successful on first attempt (ONE), successful within three attempts (THREE), successful in more than three attempts (MANY), attempted but all attempts were unsuccessful (FAILED), and did not discover (UNKN).

Experimental Design

We used a between-participants design; each participant saw one condition and all four tasks. Tasks were presented in the same order for all participants.

Procedure

Similar to Experiment 1, participants were read an introduction explaining that they would be asked to create a series of diagrams using a program. As before, participants were asked to think-aloud. Participants were told to go at their "normal working pace" and to press the Start button to begin a task, and the Stop button once they had finished a task. Participants were told that the diagrams they created did not have to be "100% perfect" but to "do as well as you can." They were also told that they did not have to match the color and exact alignment of the target diagram.

To test approachability, it was important to give users no training, warm-up tasks, or introduction. Participants were *not* introduced to the concept of a gesture to ensure that their *a priori* knowledge would reflect what end users might have in the real world, as most users have no experience with pen gestures. However, believing that users of a specific application would probably have heard, at the very least, a minimal amount of high-level information about the application, we gave each participant one sentence of background, "You will be using Lineogrammer which has powerful tools for creating and editing diagrams."

Participants were told that the experiment moderator would not be able to answer questions about the software itself, but could clarify a given task if it was unclear. Experiment moderator was only permitted to answer specific questions directly related to the task. Nothing related to the capabilities, features, or use of the application was mentioned at any time during the experiment.

Results

The discovery rate and overall coverage data lets us examine the effectiveness of the four experimental conditions in assisting users in successfully finding gestures and invoking them. Figure 7 summarizes this data. A one way ANOVA showed significant differences for both overall coverage rate ($F_{3,40} = 8.8$, $p < 0.05$) and discovery rate ($F_{3,40} = 3.95$, $p < 0.05$). To further explore each metric we conducted a post-hoc analysis, performing pair wise comparisons on the four conditions using independent sample 2-tailed t-tests. To control for the chance of Type I errors, we used Holm's sequential Bonferroni adjustment [11] with 3 comparisons, $\alpha = 0.05$ for each. GBAR provided a discovery rate of 90.9% which is significantly higher than CRIB ($t_{20} = -4.05$, $p < 0.0167$) at 75.1% and ANIM ($t_{16.524} = -2.81$, $p < 0.025$) [1] at 77.0%. However, no significance was found for discovery rate between GBAR and EXPLOR ($t_{20} = -1.86$, $p = 0.078$) at 81.8%. This indicates users were much better at discovering the gestures needed to optimally perform the given tasks using GBAR and EXPLOR. For correctly performing gestures, GBAR had an overall coverage of 87.6%, which was significantly higher than CRIB ($t_{20} = -5.41$, $p < 0.0167$) at 58.4%, ANIM ($t_{20} = -5.51$, $p < 0.025$) at 60.3%, and EXPLOR ($t_{20} = -2.17$, $p < 0.05$) at 72.7%. Although GBAR and EXPLOR are better suited for gesture discovery than CRIB and ANIM, GBAR is best in terms of overall performance. Surprisingly, there was little change in mean overall coverage between CRIB and ANIM.

[1] Levene's test for equality of variance was significant for this comparison yielding a correction in the degrees of freedom.

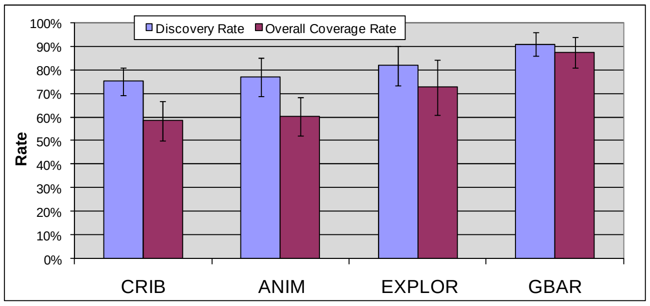


Figure 7. Mean discovery rate and mean overall coverage rate by condition (greater is better); 95% CI shown.

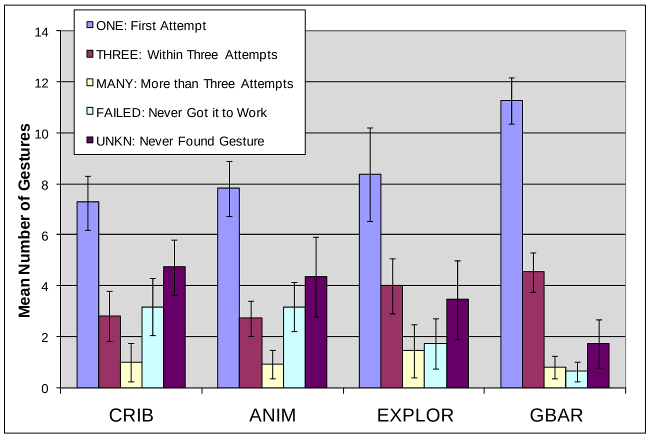


Figure 8. Histogram of mean gesture attempts classified into performance category by condition (ONE is best, THREE is better; MANY, FAILED, UNKN are worst); 95% CI shown.

We collected the performance category data in an effort to break down overall coverage to see how many attempts it took to successfully invoke gestures. A summary of this data is shown in Figure 8. We were most interested in ONE and FAILED because being able to complete a gesture on the first attempt or not at all is important to the overall usability of a gestural system; using a one way ANOVA, significant differences were found for ONE ($F_{3,40} = 7.51$, $p < 0.05$) and FAILED ($F_{3,40} = 7.02$, $p < 0.05$). We also conducted a post-hoc analysis using the same criteria as the overall coverage and discovery rate metrics. GBAR participants performed gestures correctly on their first attempt with a mean of 11.27, which is significantly higher than CRIB's 7.26 mean ($t_{20} = -5.59$, $p < 0.0167$), ANIM's 7.82 mean ($t_{20} = -4.77$, $p < 0.025$), and EXPLOR's 8.36 mean ($t_{20} = -2.17$, $p<0.05$). For FAILED, GBAR, with a mean of 0.63, significantly reduced the number of times a participant could not get a gesture to work over CRIB ($t_{20} = 4.21$,

$p < 0.0167$) and ANIM ($t_{20} = 4.71, p < 0.025$), both with a mean of 3.18. However, no significance was found for FAILED between GBAR and EXPLOR ($t_{20} = 2.00$, $p = 0.059$), with a mean of 1.73. GBAR's and EXPLOR's provision of a practice area might explain this result. There was no significant difference in average practice area uses between EXPLOR and GBAR ($t_{20} = 1.72$, $p = 0.101$).

DISCUSSION

The results from Experiment 2 support our core hypothesis, as GBAR users were able to successfully execute nearly 90% of the gestures needed to optimally perform the tasks they were given – a value which was significantly higher than the other conditions. This suggests that, with a well designed gesture set, users could achieve over a 90% mean coverage rate of an application's gestural commands, which we believe is sufficient to support walk-up-and-use experiences for commercial software. Experiment 1 also showed that users overwhelmingly chose GBAR over the other conditions for finding and learning gesture commands.

Figure 7 shows that for GBAR, the mean overall coverage rate is closer to the mean discovery rate than in EXPLOR which implies that GBAR users are learning a higher percentage of the commands they discover than in EXPLOR. This is surprising as there is no additional information in GBAR to aid the gesture learning–in fact there is less, since EXPLOR has gesture demonstration icons in the crib sheet. Anecdotally, we noticed that GBAR users appeared more comfortable and confident perhaps as a result of using an interface which is familiar. This confidence may have led them to attend more closely to the Gesture Explorer content, resulting in more successful learning.

One can think of performing a gesture correctly on the first attempt as an ideal experience, whereas one can think of failing to perform a gesture correctly at all – probably after a number of failed attempts – as a poor experience. Based on this framework, one can say that users of GestureBar had a better experience as they successfully performed significantly more gestures on their first try, and failed significantly fewer gestures than CRIB or ANIM, and there was no significant difference in failures from EXPLOR. The category data also supports this as in Figure 8 one can see that with GestureBar a high level trend appears to be away from the worst categories (UNKN, FAILED and MANY) toward the better categories (ONE, THREE).

The difference in performance between CRIB and ANIM was minimal, suggesting that animation alone does not necessarily improve performance. We suspect that the speed of the animation may have de-emphasized geometric nuances. One participant said of ANIM, "learning how to make certain shapes was vague and a bit difficult," which was mirrored by several other participants. Interestingly, animation did appear to help with the disclosure of stroke direction. We noticed that a number of users drew strokes in the reverse direction, resulting in recognition failures, but only when using CRIB. Another problem which affected both CRIB and ANIM was initial uncertainty about the gesture interaction model, which in some cases persisted throughout the participant's use of the system. In some cases, users would tap repeatedly on the crib sheet and then tap on objects in the document in an attempt to execute commands. This behavior was largely absent in EXPLOR and completely absent in GBAR. This suggests users were driven to find an outlet for their habitual "point-and-click" behavior which was most naturally accommodated by GBAR.

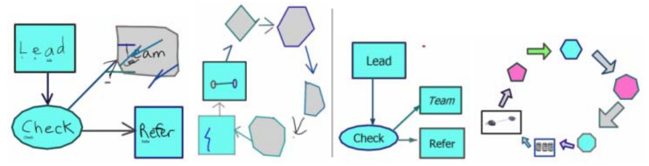


Figure 9. Sample experiment 2 diagrams (left to right): CRIB task 1; CRIB task 2; GBAR task 1; GBAR task 2.

A related observation is that for all conditions, some users felt that they needed to click on a tool before performing a gesture. CRIB and ANIM users complained about having to do this, but surprisingly EXPLOR and GBAR users did not. We believe this reflects an overall sense of confusion and dissatisfaction with CRIB and ANIM. However, at least one user in each condition commented that it was frustrating to find a command, such as Undo, and then have to learn a gesture in order to perform it (as opposed to just clicking a button). It is not clear how severe a problem this is since Grossman's related work [7] observed that users had a positive perception of his disabled technique in which menu items were disabled to force usage of hot-keys.

An intuitive way to visualize the difference in performance between GBAR and CRIB is to review sample diagrams made by users (see Figure 9). These sample diagrams clearly show a significant difference in terms of real productivity between the average coverage rates of 58.37% for CRIB and 87.56% for GBAR. GBAR also had a higher mean discovery rate than the crib sheet designs which we largely attribute to its less cluttered layout that makes higher level categorizations stand out (users commented that they liked the command categorizations in GBAR even though the same categorizations, including headings, were present in CRIB as well). However, the higher discovery rate for EXPLOR over the other crib sheet conditions, suggests that finding high-value information during one search has the added benefit of reinforcing future searching behavior.

In almost all cases, participants using ANIM and EXPLOR opened the animation/gesture explorer for each gesture they wanted to learn; however, in a handful of cases, users opted to simply read the crib sheet icon. In two of these cases, the participant failed the gesture attempt and then opened the gesture explorer before continuing. It is possible that over time, some users may become comfortable with the concept of a gesture and find referencing a crib sheet more convenient, perhaps as an optional power-user feature. It is also possible that after a sufficient introductory period, users might switch to an "expert mode" to save screen space.

FUTURE WORK

The fundamental nature of gesture UIs implies that, at times, gestures will be misrecognized. We are thus interested in techniques which either reduce the likelihood of errors or at least make it easier to recover from them. One approach is to encourage users to make more use of the Practice Area. This might happen as a byproduct of just improving the feedback of the Practice Area, perhaps by integrating the disclosure techniques used in OctoPocus [3], or by incorporating explicit textual answers to the question "why didn't my gesture work?" as implemented in Microsoft Vista's Flick tutorial. Alternatively, we might exploit workflow context, such as knowledge of which gestures had recently been encountered in the GestureBar, either to improve recognition tolerance or to provide suggestive recovery or performance feedback after ensuing failed attempts. GestureBar could also be applied to multi-touch and speech UIs.

CONCLUSION

We have presented GestureBar, a familiar-looking toolbar UI, which, instead of executing a command when clicked, richly discloses the gestural interactions needed to execute the command. Thus, unlike crib sheets which present an entirely unfamiliar UI metaphor, GestureBar users naturally and necessarily discover gestural interaction as a byproduct of approaching the GestureBar as if it were a conventional toolbar. A quantitative, ecologically valid study of user performance supports our motivating hypothesis that even novices, with no prior exposure to gestural interaction, can immediately and successfully use a gestural application without the need for human assistance or up-front training. A qualitative study of user preferences shows that GestureBar is preferred over the other crib sheet-based conditions for finding, and learning commands. Finally, GestureBar presents no barriers to widespread adoption in terms of required recognition technology or gesture set constraints, and it can be easily unified with standard toolbar elements.

ACKNOWLEDGMENTS

A special thanks to Andries van Dam, Ken Hinckley, and Steven P. Reiss for their advice and insight, and to IARPA and the Microsoft Center for Research on Pen-Centric Computing at Brown University for their support.

REFERENCES

1. Alvarado, C. Sketch Recognition User Interfaces: Guidelines for Design and Development. *In Proc. of AAAI Fall Symposium on Intelligent Pen-based Interfaces*, (2004).

2. Alvarado, C. and Davis, R. SketchREAD: a multi-domain sketch recognition engine. *In Proc. of UIST'04*, 23-32.

3. Bau, O., and Mackay, W. OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets. *UIST'08*, 37-46.

4. Buxton, W. Chunking and phrasing and the design of human-computer dialogues. *IFIP World Computer Congress'86*, 475-480.

5. Buxton, W., Fiume, E., Hill, R., Lee, A., Woo, C. Continuous Hand-Gesture Driven Input. *Graphics Interface '83*, 191-195.

6. Forsberg, A, Holden, L., Miller, T., and Zeleznik, R. *The Music Notepad*, Brown University (2005).

7. Grossman, T., Dragicevic, P. and Balakrishnan, R. Strategies for Accelerating On-line Learning of Hotkeys. *CHI'07*, 137-144.

8. Grossman, T., Hinckley, K., Baudisch, P., Agrawala, M., Balakrishnan, R. Hover Widgets: Using the Tracking State to Extend the Capabilities of Pen-Operated Devices. *CHI'06*, 861-870.

9. Hinckley, K., Baudisch, P., Ramos, G., Guimbretiere, F. Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli. *In Proc. of CHI'05*, 453-460.

10. Hinckley, K., Zhao, S., Sarin, R., Baudisch, P., Cutrell, Ed., Shilman, M., Tan, D. InkSeine: In Situ Search for Active Note Taking. *In Proc. of CHI* (2007), 251-260.

11. Holm, S. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6 (1979), 60-65.

12. Hong, J. and Landay, J.. SATIN: A Toolkit for Informal Ink-Based Applications. *In Proc. of UIST* (2000), 63-72.

13. Hse, H. and Newton, A. Recognition and beautification of multi-stroke symbols in digital ink. *Computers & Graphics*, 29, 4 (August 2005), 533-546.

14. Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H. Interactive Beautification: A Technique for Rapid Geometric Design. *In Proc. of UIST* (2007), 105-114.

15. Kurtenbach, G., Moran, T. P. and Buxton, W. Contextual Animation of Gestural Commands. *Graphics Interface '94*, 83-90.

16. Long, C., Landay, J., Rowe, L., and Michiels, J. Visual Similarity of Pen Gestures. *In Proc. of CHI'00*, 360-367.

17. Mankoff, J., Hudson, S. and Abowd, G. Providing Integrated Toolkit-level Suport for Ambiguity in Recognition-based Interfaces. *CHI'00*, 368-375.

18. Microsoft, Inc. *Office 2007*. 2006.

19. Microsoft, Inc. *Windows Vista*. 2006.

20. Mouse Gestures Add-on for the Mozilla Firefox Web Browser. http://www.mousegestures.org/.

21. Palm, Inc. *Graffiti character recognizer.*

22. Polson, P. and Lewis, C. Theory-Based Design for Easily Learned Interfaces. *Human-Computer Interaction*, 5, 2 (June 1990), 191-220.

23. Rubine, D. Specifying Gestures by Example. *In Proc. of SIGGRAPH'91*, 329-337.

24. Tapia, M., and Kurtenbach, G. Some Design Refinements and Principles on the Appearance and Behavior of Marking Menus. *UIST'95*, 189-195.25.

25. Wobbrock, J., Wilson, A., and Li, Y. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. *In Proc. of UIST* (2007), 159-168.

26. Zeleznik, R., Bragdon, A., Liu, C., and Forsberg, A. Lineogrammer: Creating Diagrams by Drawing. *UIST '08*, 161-170.

27. Zeleznik, R., and Miller, T. Fluid Inking: Augmenting the Medium of Free-Form Inking with Gestures. *In Proceedings of Graphics Interface* (2006), 155-162.

28. Zhao, S., Agrawala, M. and Hinckley, K. Zone and Polygon Menus: Using Relative Position to Increase the Breadth of Multi-stroke Marking Menus. *CHI* (2006), 1077-1086.