Volume 13, (1994) number 5 pp. 305–314

COMPUTER GRAPHICS *forum*

Contextual Animation of Gestural Commands

G. Kurtenbach*, T. P. Moran*, and W. Buxton‡†

* Xerox Palo Alto Research Center, Palo Alto, California, USA
‡University of Toronto, Toronto, Ontario, Canada

Abstract

Drawing a mark can be an efficient command input technique when using a pen-based computer. However, marks are not intrinsically self-explanatory as are other interactive techniques such as buttons and menus. We present design principles for interaction mechanisms which make marks self explanatory for novices but still allow experts to use efficient command marks. The key notion is that use of the explanation mechanism physically trains a novice to use the efficient command marks. Two novel interaction mechanisms we have developed using these principles are presented.

Keywords: Pen input; Gestures; Marks; Animation

1. Introduction

It is a very common belief that pen-based computers will be easy to use because "they operate like pen and paper". Essentially, pen-based computers try to take advantage of a user's existing drawing and handwriting skills. The marks made with an electronic pen can be recorded as "ink", which is coming to be accepted as new basic kind of data. In addition to creating material to be seen and read, people also use marks to designate actions on the material – from informal marks such as scratch-outs and arrows to more formal notations such as proof-reading marks. Electronically-produced marks can also be interpreted, and these marks are usually called *gestures* in the user-interface community. Gestures are thus iconic commands from the user to the system.

It would seem that gestures would be easy to learn and use. However, one needs only to use any of the current crop of pen-based computers to experience serious difficulties. Recently, we assessed a new, sophisticated note-taking application that was touted as being natural and easy to use (and, in the press, as a real breakthrough in pen computing). When we sat down to learn and use the system, we expected this to be easy. After only a short while we found ourselves asking questions like: "What gesture do I make to undo something?""Are there commands available with gestures that are not in the menus?" "Why isn't it interpreting my X-gesture as a delete gesture?" "Does it understand the standard proof-reading marks?"

This situation is reminiscent of old-fashioned textual command language interfaces, such as the UNIX shell or MS-DOS, where the user is confronted with analogous questions. Thus, the issues behind the questions seem to be general to all command languages, be they textual or iconic:

Functionality - What functions does the system provide (in the form of commands)?

Naming - Given a function, what is the name or shape of the command (so that it can be issued)?

Context - Given a command, when and where in the system is it available to be used?

Method - How are the various arguments and parameters of a command specified (so that it can be applied to specific material in a specific way)?

There are several different strategies that the user can employ to answer these questions. Let us consider three : training, guessing, and learning-by-doing.

Training Strategy - The user can set aside a chunk of time to learn the system–take a course, read the manual,

† G. Kurtenbach and W. Buxton are now at Alias Research Inc. Toronto, Ontario, Canada.

follow an on-line tutorial, etc. One problem with this strategy is that it is not tied to any particular task the user needs to do. During training, the user, in effect, memorizes the system ahead of time. Later, when it is time to do a particular task, the user may have forgotten many of the crucial details and will end up posing the same questions anyway. The goal of most pen-based systems is to be "natural" so as not to require up-front training, the ideal being that one can just "walk up and use" them. Therefore, we want to minimize the need for training.

Guessing Strategy - The user can forego training and just guess how to issue commands. This depends on the commands being mnemonic. For verbal commands, it has been shown that mnemonics are unreliable; command naming behaviour of individuals is extremely variable[1]. But gestures are supposed to be intuitive and/or familiar. Many researchers have argued that users commonly agree on certain gestures for certain operations[2-4]. However, beyond a small set of common operations (e.g. select, delete, move), there are few common conventions (mainly because gestural systems are so new). Thus, guessing by itself is inadequate.

Learning-While-Doing Strategy - A broader strategy is for the user to seek help in various ways *while doing particular tasks* and, *in the process,* learn more and more about the system. Thus the need for (and time taken in) seeking help is continually reduced. The critical thing to make this work is to minimize the amount of attention the user has to divert from the performance of the task in order to seek help (training and guessing are at the two extremes).

We can view many interface techniques of modern graphical user interfaces as supporting a learning-while-doing strategy. Menus of commands and panels of buttons and icons tell the user what functions are available and directly provide the means to invoke them. They allow users to recognize functions rather than having to recall them from memory. Two examples are menus and dialogue boxes. Menus that pop up when certain objects are selected and pull-down menus with greyed out items show users the context in which commands are available. Dialogue boxes give users simple methods for specifying parameters to commands.

What we propose is to extend these graphical user interface techniques with two specific goals in mind: (1) supporting the process of learning-while-doing and (2) dealing with the particular features of gestural commands. A couple of examples: We will consider techniques for inducing rehearsal, which is important to amplify the learning process. Gestures have the feature that they are drawn *within* the materials they are operating on (whereas textual commands, including menus, are issued from outside of the materials). Thus we have to provide guidance for how to draw gestures within the spatial context of the current materials. In this paper we define three user-interface design principles to support interactively learning and using gestures. We then describe two interaction techniques we have developed based on these design principles. The first technique supports learning and using the subclass of zig-zag-shaped gestures. The second technique deals with the general case of learning and using arbitrary-shaped gestures.

2. Design Principles

The three design principles to support learning and using gestures are *revelation, guidance,* and *rehearsal.* Other researchers have described similar general principles[5,6], and many systems have interactions which follow some of these general principles. Our definitions are oriented to apply the principles to gestures.

Revelation - The system should interactively reveal information about what commands are available and how to invoke them.

Gestures are not revealed because the user must recall them from memory. Menus and buttons, however, reveal the function and names of commands. They do not reveal the method for issuing the command. What menu systems do is to provide a common set of general methods (such as pointing, dragging, double clicking), which must be learned *a priori.* The Macintosh computer, for example, uses this technique. The intention is that with this small set of skills a user can start interactively exploring and learning about the remainder of the system.

The interaction techniques described in this paper use this type of design. A user must be informed, *a priori,* that in order to reveal the commands associated with an object the pen must be pressed over an object and held still for a fraction of second. We call this "press and wait for more information". Once users know this, they can get further instructions interactively from the system. This allows users to interactively learn about what functions can be applied to various displayed objects by *pressing and waiting* on the objects for menus.

Guidance - The way in which revelation occurs should guide a user through the method for specifying the complete command in any specific situation.

An example is selection from a hierarchic menu. In this case, selecting an item guides a user to the next menu. The critical point in these systems is that getting guidance on how to specify a command does not interrupt the specification process. On the other hand, a system like the on-line manual pages in UNIX violates the principle of guidance, because the user must terminate or at least suspend the act of specifying the command in order to get help.

Rehearsal - The way guidance is provided should require a physical rehearsal of the way an expert would issue the command.

The goal of rehearsal is to develop expert skills in a novice, in order to support the efficient transition from novice to expert performance. Many interaction techniques support rehearsal. When the action of the novice and the expert are the same for a particular function, we can say that rehearsal takes place. For example, novices may draw lines, move icons, or select from menus using the same actions as an expert when there is one and only one way of issuing the command. In many cases, the single way of issuing the command may be suitable for both the novice and expert.

There are also many situations, however, where a single method for invoking a command is not sufficient. The popularity of "accelerator techniques"is proof of this. Typically, interfaces provide two modes of operation. The first mode, designed for novices, provides revelation. Conventional menu-driven interactions are an example of this. The revealing component of this mode is emphasized over efficiency of interaction, because novices are more concerned with how to do things rather than how quickly things can be done. The second mode, designed for experts, typically allows terse, non-prompted interactions. Command-line interfaces and accelerator keys are examples of this mode. However, usually there is a dramatic difference between novice and expert behaviour at the level of physical action. For example, a novice uses the mouse to select from a menu whereas an expert presses an accelerator key. Thus, in these cases, novice actions are not a rehearsal for expert performance.

It is critical that rehearsal be unavoidable. For example, the Macintosh supports novices by providing menus and supports experts by providing menu accelerator keys. The transition between novice and user is supported by the user being reminded of the keystrokes associated with menu items every time a menu is displayed. This is done by having the names of the accelerator keys appear next to menu items in the menu. However, actually using an accelerator key is avoidable. The user can always just select from the menu. Furthermore, this is easiest because the user is already displaying the menu. The end result is that accelerator keys are sometimes not used even after extensive exposure to the menu. Our principle of rehearsal is intended to remedy these situations.

The intention of the three design principles is to reduce this discrepancy in action without reducing the efficiency of the expert and ease of learning for the novice. The basic actions of the novice and expert should be the same. It is hoped that, as novice performance develops, the skills that lead to expert performance will develop in a smooth and direct manner. We next describe two interaction techniques that apply the design principles to gestures.

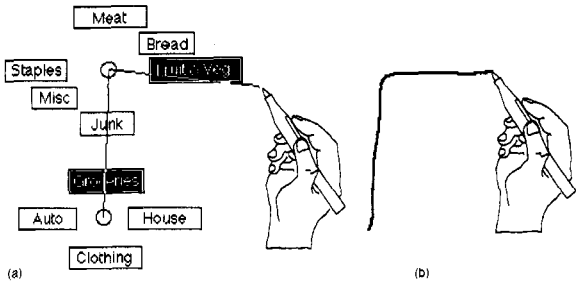

Figure 1: *Marking menus permit two different ways to select menu items. Using method (a), hierarchic radial menus can be sequentially displayed and selections made. Method (b) uses a mark (gesture) to make the same selection.*

3. Marking Menus

Rather than trying to initially solve the general problem of providing revelation, guidance and rehearsal for any type of gesture, we asked ourselves if there were subclasses of gestures which simplified the problem.

With this goal in mind we developed an interaction technique called *marking menus.* Marking menus provide revelation, guidance, and rehearsal for zig-zag types of gestures. This is done by integrating pop-up radial menus and zig-zag gestures. In effect, zig-zag gestures are the by-product of selection from radial menus. This works as follows: A novice user presses down on the screen with the pen and waits for a short interval of time (approximately 1/3 second). A radial menu7,8 then appears directly under the tip of the pen. A user then highlights an item by keeping the pen pressed and making a stroke towards the desired item. If the item has no sub-menu, the item can be selected by lifting the pen. If the item does have a sub-menu, it is displayed. The user then continues, selecting from the newly displayed sub-menu. Figure 1 (a) shows an example. Lifting the pen will cause the current series of highlighted items to be selected. The menus are then removed from the screen. At any time a user can indicate "no selection" by moving the pen back to the center of the menu before lifting, or change the selection by moving the pen to highlight another item before lifting. A user can also "back-up" to a previous menu by pointing to its center.

The other (and faster) way to make a selection is by drawing a gesture. A gesture can be drawn by pressing the pen down and immediately moving. The shape of the gesture dictates the particular series of items selected from the menu hierarchy. Figure 1 (b) shows an example.

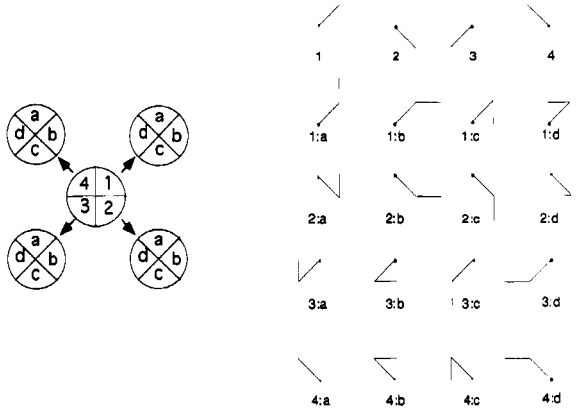

Figure 2: *An example of a radial menu hierarchy and the marks that select from it. Each item in the numeric menu has a submenu consisting of the items a, b, c and d. A mark's label indicates the menu items it selects. A dot indicates the starting point of a mark.*

In effect, the menu reveals the commands associated with a vocabulary of zig-zag gestures. Figure 2 shows an example of a zig-zag gesture vocabulary and the menu that reveals them.

Marking menus adhere to the design principles as follows: Revelation is provided by the pop-up menu (the novice can see what commands are available). Guidance is provided by the system giving the user feedback and additional menu items as the menu is traversed. Rehearsal is provided by the physical movement involved in selecting an item from the menu being identical to the movement required to make the gesture corresponding to that item.

We have extensively user tested marking menus and have found that they are used as designed. Novices pop-up the menus but with experience learn to use the gesture (i.e., they become experts). Drawing a gesture has been show to be dramatically faster than traditional menu selection techniques. See Kurtenbach'swork9 for an in-depth analysis of marking menus.

4. The Crib/Sheet Animator

Can an interaction technique similar to marking menus be designed for other types of gestures? In other words, can the design principles be applied to the general case? We refer to these other kinds of gestures as *iconic gestures* (although the meanings of these gestures may not be strictly based on iconic shape) and we refer to marking menu's zig-zag gestures as *menu gestures.* Thus the question is: can revelation, guidance and rehearsal be provided for iconic gestures?

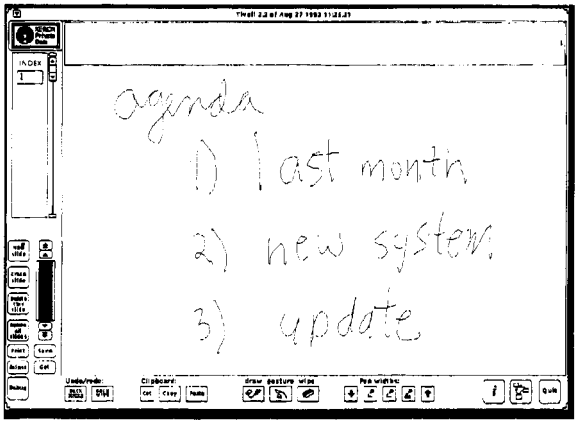

Figure 3: *An application called Tivoli, running on Liveboard, emulates a whiteboard but also allows drawings to be edited, saved and restored.*

In order to investigate this question we decided to take an existing pen-based application that used iconic gestures and attempted to design an interaction mechanism that would provide revelation, guidance and rehearsal for those gestures. The test bed for this design experiment was an electronic whiteboard application called *Tivoli10.* Tivoli is intended to be used in collaborative meeting situations, much in the same way that a traditional whiteboard is used. Tivoli runs on a large vertical display, called *Liveboard,* that can be written on with an electronic pen. Much like a whiteboard, several people can stand in front of a Liveboard and write, erase, gesture at, and discuss hand drawn items. Handwriting and drawings also can be edited by a combination of direct manipulation commands (i.e. buttons, menus, etc.) and iconic gestures. Figure 3 shows Tivoli and Figure 4 shows the types of iconic gestures used.

4.1. Problems with the Marking Menu Approach

Overlap - Suppose we strictly applied the marking menu design to these gestures. Essentially, a marking menu displays the various ways a user could move the pen to issue a command. Figure 5 shows the result of applying this approach to some of the gestures in Figure 4. When a user presses the pen at a location, the system displays the various ways a user could move the pen by displaying example gestures. As Figure 5 shows, gestures overlap and can cause confusion. Part of the problem is that iconic gestures are not suitable for displaying in this manner. Menu gestures, however, are suitable because of their directional and segmented nature. Only the first segment of the zig-zag gesture needs to be displayed. The

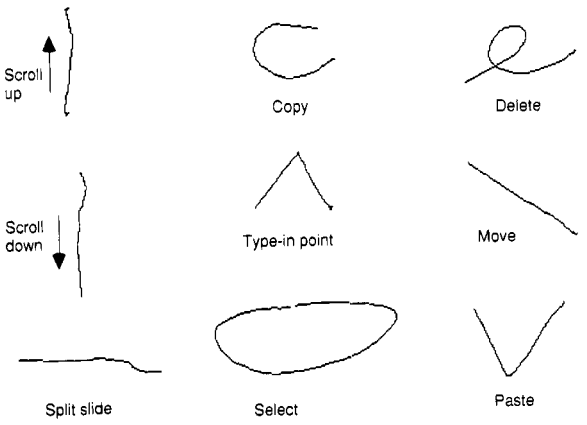

Figure 4: *The basic gestures used in Tivoli.*

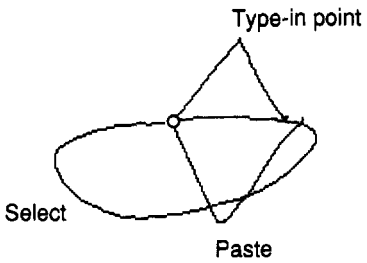

Figure 5: *Overlap causes confusion when using the marking menu approach to reveal other types of gestures. Here we display the commands available when starting a gesture from a clear spot in the drawing region of Tivoli.*

remaining segments of the gesture can be incrementally displayed as the menu is traversed.

Not enough information - Another problem with a display like Figure 5 is that it gives little contextual information. For example, the important thing about the "Select" gesture is that it should encircle objects and the shape of the circle can vary. This type of information is not shown in Figure 5.

The meaning of several iconic gestures in Tivoli is determined not only by the shape of the gesture but also by the context in which the gesture is made. For example, a straight line over a bullet-point moves an item in a bullet-point list, while a straight line in a margin scrolls the drawing area. These types of inconsistencies can potentially confuse the user. To avoid these problems, we wanted to provide context sensitive information about which gestures a user can make over what objects. Informally, we wanted a user to be able to answer the question: "what gestures can I draw on this object or location?". Since marking menus are sensitive to context (i.e., the contents of a menu may vary depending on where it is popped up), we hoped that some similar mechanism could be designed for iconic gestures in Tivoli.

For gesture sets in general, besides Tivoli's iconic gesture set and the marking menu gesture set, the following characteristics may contribute to a gesture's meaning and this type of information therefore needs to be revealed:

Shape - This is the case where a particular shape is an icon for a certain command. For example, the "pigtail" shape is an icon for the delete command.

Direction - Sometimes the direction of a gesture affects its meaning. For example an up-stroke means "scroll up" while a down-stroke means "scroll down". The shape of the gesture is basically the same but the direction or orientation of the gesture has meaning.

Location of features - The location of features of a gesture can affect its meaning. For example, the summit of the "Type-in" point gesture, shown in Figure 5, determines the exact placement of the text cursor.

Dynamics of drawing - How a gesture is drawn can affect its meaning. For example, a flick could mean "scroll to the end of document", while a slow up-stroke could mean "scroll to the next page".

4.2. Solutions

Crib-sheets - Interactive crib-sheets reveal gestures without the overlap problem. When the user requires help, a crib-sheet can be popped up which shows the available gestures and what they mean. The user can then dismiss the crib-sheet and make a gesture. Other systems have used mechanisms that are similar to crib-sheets (e.g., *XButtons11* and Microsoft's *Windows for Pen Computing).* Crib-sheets can be as succinct as a simple list of named gestures or as elaborate as multi-page explanations of the gestures in great detail. Thus a crib-sheet could contain complete information on all the characteristics of a gesture. However, since crib-sheets are for reminding and guidance, they are usually succinct.

Figure 6 shows the crib-sheet technique we designed for Tivoli. The design works as follows. Similar to a marking menu, if one doesn't know what gestures can be applied to a certain object or location on the screen, one presses-and-waits over the object for more information, rather than drawing a gesture. At this point, rather than a menu popping up as in the marking menu case, a crib-sheet is displayed. The crib-sheet displays the names of the functions that are applicable to the object or location, and example gestures. If this is enough information, a user can draw one of the gestures in the crib-sheet (or take any other action) and the crib-sheet automatically disappears. If the pen is released without

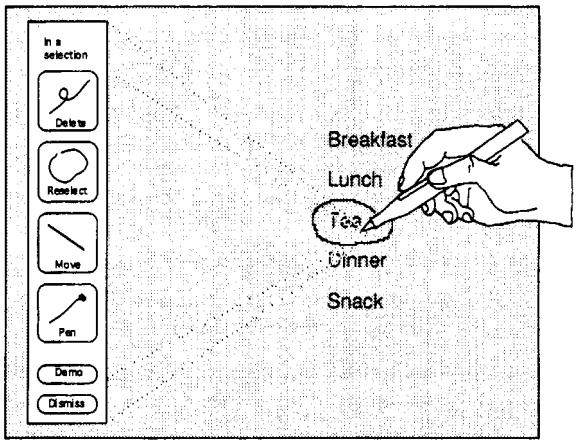

Figure 6: *Revealing iconic gestures in Tivoli: The user has selected the word "Tea" by circling it. To reveal what functions can be applied to the selection, the user presses-and-waits within the selection loop. A crib-sheet pops up indicating the context ("In a selection") and the available functions and their associated gestures.*

drawing a gesture, the crib-sheet remains displayed until the next occurrence of a pen press followed by a pen release or a press-and-wait event.

This design has several important features which distinguish it from a pop-up menu. First, the system displays the crib-sheet some distance away from the pen tip so that the crib-sheet does not occlude the context. This leaves room for a user to draw a gesture. Second, a user must draw a gesture to invoke a command. For example, a user cannot select the delete button to perform a deletion. The user must draw a delete gesture to perform a deletion. Finally, the significance of the location of the pen tip is displayed at the top of the crib-sheet (i.e., in Figure 6 "In a selection" is displayed at the top of the crib-sheet). This is useful for revealing the meaning of different locations and objects on the screen.

This design obeys the principles of revelation, guidance, and rehearsal. The crib-sheet provides revelation, and a user can use the examples as guidance when drawing. Rehearsal is enforced because a user must draw a gesture to invoke a command rather than pressing on a crib-sheet item.

Animated, annotated demonstrations - While the crib-sheet does reveal contextual information about gestures, it still lacks certain types of information. For example, one static example of a gesture relays little information about variations and features of a gesture. It has been shown that people need good examples to help visualize procedures[12]. Ideally a demonstration of the gesture in context should be provided, similar to what one receives when an expert user demonstrates a command. The tutorial program in *Windows for Pen Computing* works like this. In the tutorial, a user is shown how gestures are made by animated examples.

The idea of animated help in direct manipulation interfaces is not new[13–15]. Our system is unique in that it uses animated help for pen-based interactions.

The examples in the crib-sheet could be animated to show how to draw a gesture, variations on a gesture, and the various features of a gesture. However, crib-sheets illustrate gestures outside of the context of the material that the user is working on, and this can make it difficult to see how the gesture applies to the context. Marking menus, on the other hand, have the advantage of showing the available gestures directly on top of the object being worked on.

To solve this problem we extended the function of the crib-sheet by adding animations of gestures which take place in context. If the crib-sheet does not provide sufficient information, a demonstration of a gesture can be triggered by pressing the "demo" button on the crib-sheet. The demonstration of the gesture begins at the location originally pressed. The demonstration is an animation of the drawing of the gesture which is accompanied by text describing the special features of the gesture (see Figure 7).

There are several important aspects to this design:

- Gestures are shown in context. The animation of the gesture is full size, and emanates from the exact location originally pressed on by the user.
- Variations in gestures are shown by multiple animations. Usually, two examples are enough.
- Information about features or semantics of a gesture is provided by annotations. (e.g., in Figure 7 "A pigtail deletes the *selected objects.*").In addition, features of the application can be displayed. For example, in Tivoli scrolling gestures can only be drawn in the margins of the drawing area, but the borders of the margins are not visible (this was done to keep the drawing area uncluttered). In situations like this, the animation can display these features to clarify matters. Annotations appear in sequence during a gesture's animation, and they are timed to remain on the display long enough for the user to read them.
- Animation can be controlled. A long series of animations takes quite a bit of time and this can be tedious for the user. By pressing a button in the crib-sheet, individual animations of the gestures can be started or stopped. Pressing the "Dismiss" button will stop the animation and removes the crib-sheet. The animation will freeze if a user begins drawing a gesture (so a user can trace the animated gesture). As in the case of the crib-sheet by itself, the moment a user completes

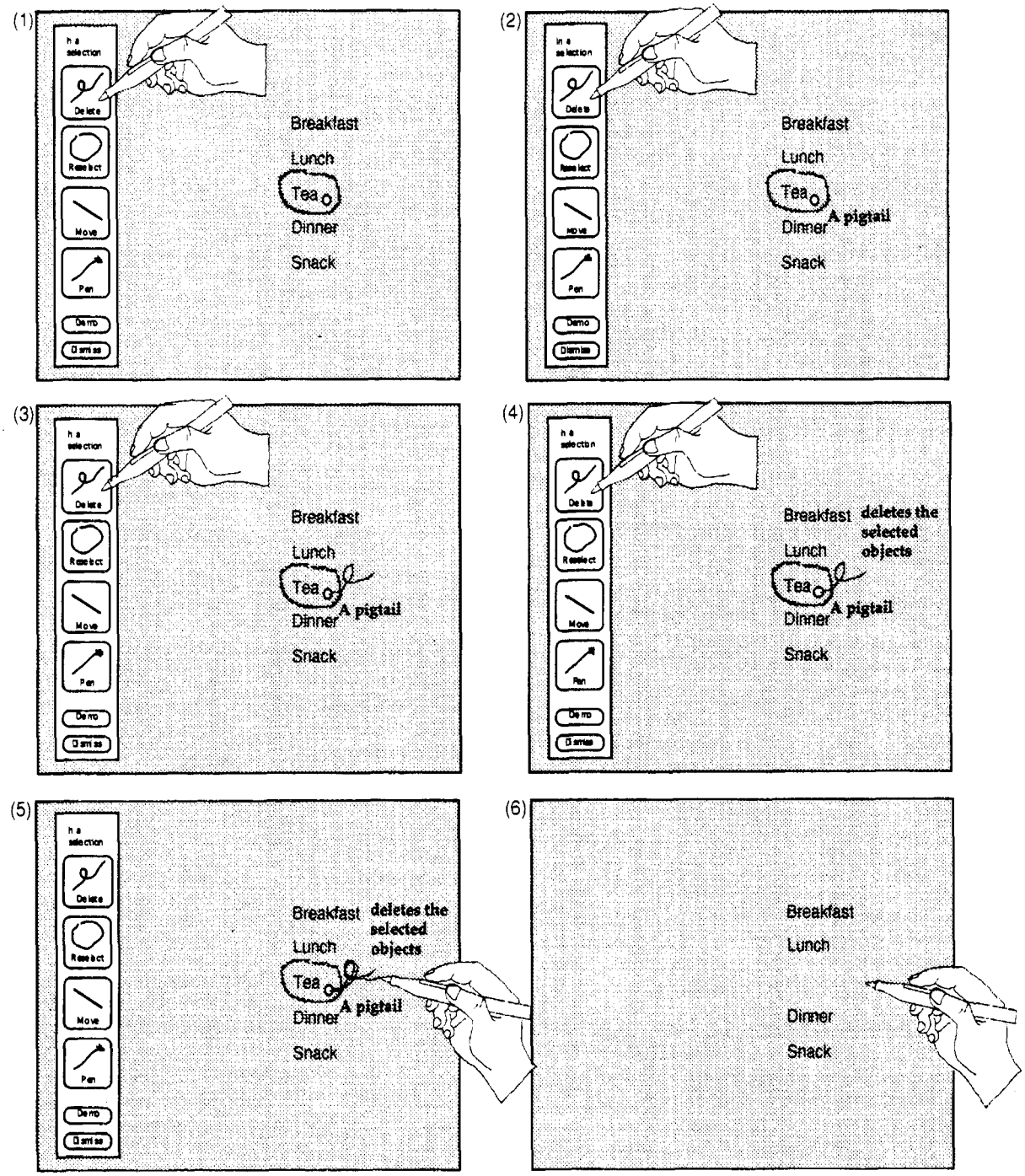

Figure 7: *A demonstration of a particular function can be attained by pressing its icon. In (1) the user presses on the delete icon for more information. This triggers an animated demonstration of the gesture with text annotation to explain its features. This is shown in (2), (3) and (4). In (5), the user traces along the example gesture to invoke the function. When the pen is lifted, the action for the gesture is carried out, and the crib-sheet and animation disappear (shown in (6)).*

a gesture, the crib-sheet is removed and the animation terminates.

- The user is not required to make a gesture from the crib-sheet. The user is free to perform any gesture at any location on the screen while the animation is running. The user can also choose to not draw a gesture by tapping the pen against the screen. This removes the animation and crib-sheet.

4.3. Implementation

Our crib-sheet/animation is implemented so it is easy for an interface programmer to use. To produce crib-sheets and animations Tivoli interacts with a software module called the *animator*. The animator accesses a *Gesture Animation Database* (GAD). The GAD contains descriptions of examples of gestures grouped by context (see Figure 8). When the user presses-and-waits, Tivoli calls the animator with a description of the current context (e.g. "In a selection"). The animator then selects the gestures to be animated based on context, constructs and displays the crib-sheet, and animates the gestures at the user's request.

GAD is constructed by first hand-drawing the gesture examples and annotations in Tivoli, then placing these into GAD. Annotations are then labeled by where and when they should occur in the animation cycle (e.g., "start" and "end"). A gesture is a sequence of x and y coordinates which is animated by incrementally displaying the gesture. When animating a gesture the animator uses the same drawing dynamics as the original hand-drawing. In this way, dynamics of drawing can be revealed and the speed of an animation can be controlled by the constructor of the examples. The pacing of the animation of text annotations is determined by length of text: after an annotation is displayed the animator pauses for an amount of time that is proportional to the length of the text before continuing with the rest of the animation. This gives a user time to read the annotation and then watch the rest of the animation.

A key feature to this design is that extra examples of the same gesture can be placed in GAD and tagged for special purposes. If an example is tagged as "variation", the animator animates this example along with the original example of the gesture. In this way, variations on a gesture can be shown to the user. When the animator retrieves the examples from GAD, example gestures are shrunk down to be displayed in the buttons of the crib-sheet. We found it convenient to tag certain example gestures for shrinking as "icon". If no "icon" example is found, the animator shrinks the first example gesture it finds.

Multiple examples of gestures also allow the animation of gestures in constrained spaces. For example,

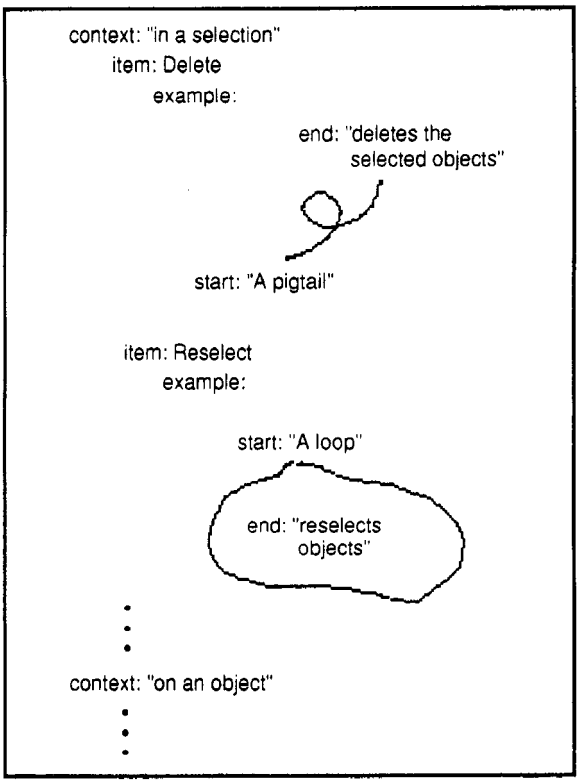

Figure 8: *An example of the structure of the Gesture Animation Database (GAD). Annotated examples of the gestures used for the crib-sheet and animations are grouped by context and function.*

assume that a user invokes the animator near the bottom of the drawing area, and that one of the possible gestures at that point is a pigtail (delete). At the bottom of the drawing area, there is no room to draw a pigtail downwards, but there is room to draw it upwards. Thus, the animator should show only pigtails that fit in this location. The solution to this problem is that when the animator retrieves examples from GAD it looks for examples that will fit in the location. Thus, GAD should be set up with several examples of each gesture, so that the animator can find an example for any location. We found as little as four different examples were sufficient. In the event that an example which fits cannot be found, the animator generates and displays a "no room message" (e.g., "not enough room to demo delete here"). This tends to only happen when there is not enough room for a user to actually draw the gesture.

Thus we could easily design examples to fit in constrained spaces by originally hand-drawing them in those spaces. For example, we drew instances of pigtails that fit at the top, bottom, left and right edge of the screen.

The animator does not have to be sophisticated at laying out the animations – the layouts are determined by the constructor of the examples. The animator need only check if an example will fit at a certain location. If it does not fit, it merely looks for another example.

4.4. Usage Experiences

The crib-sheet/animator has been used informally by several researchers at Xerox PARC. We were able to provide several examples of every gesture used in Tivoli in the GAD. Initially, we found that users did not notice the crib-sheet pop up on the left side of the display. This was because users were so close to the large display that the crib-sheet popped up outside their visual focus. We then added an animation of the crib-sheet expanding from the point at which press-and-wait occurred. This helped users notice the display of the crib-sheet.

Users were also able to make use of the crib-sheet/animator after a brief demo. We found that users explored the interface by pressing-and-waiting at different spots to see what functions where available. We also observed users tracing the animated gestures. The most common error involved a user pressing-and-waiting with the command button pressed, then releasing the button while watching the animation. The user would then trace the animated gesture without the command button being pressed (Tivoli requires a command button on the pen to be pressed for the system to interpret marks as gestures not as drawing or hand-writing). Not having the command button pressed would result in the mark being drawn but not interpreted. We feel this type of error may disappear when a user gets into the habit of holding down the command button to issue a command. It is also possible to have the system recognize this error and advise the user to press the command button.

5. Future Work

An obvious next step for future research is formal user testing of our designs. It would be optimistic of us not to expect users to have problems with our system. First, there are many details that users might trip over: are the menus and buttons labeled meaningfully? Are the press-and-wait time thresholds correct? We believe the next step in user testing would be to evaluate some of these details and refine the content of the animations.

One problem with our current implementation is that, although animations do appear in context, they do not "work with" the context. For example, the animation of a loop being drawn to select objects sometimes doesn't enclose any objects. The problem is the animator has no knowledge about the Tivoli objects underlying the animation. A more advanced version would extend the notion of parameterized gestures to allow them to utilize and manipulate Tivoli objects in the current working context. This would require a much more sophisticated architecture for the animator. A good starting point is to build on the work that Sukaviriya and Foley have done on the generation of parameterizable, context sensitive animated help for direct manipulation interface15.

6. Summary and Conclusions

Gestures have many advantages but they also have the disadvantage of not being revealing. To reveal gestures some sort of interactive mechanism must be used. We presented the design principles of revelation, guidance and rehearsal which promote the integration of the interactive mechanism and gestures. The notion is that the interactive mechanism is intended for the novice while the gestures are intended for experts. The integration of the two is intended to support the learning transition from novice to expert.

We presented two designs that follow these design principles. Marking menus integrate radial menus and zig-zag gestures and represent the application of the design principles to a subclass of gestures. The crib-sheet/animator represents the application of the design principles to any type of gesture. We have found in practice that marking menus are very effective in supporting novices and experts. The fact that the crib-sheet animator is a workable design proves that the design principles are generalizable to iconic gestures. Further design exploration and testing is warranted.

Designing a mechanism to reveal iconic gestures brings to light many issues concerning the revelation of gestures. First, revelation can occur at various levels of detail. The crib-sheet is the first level: a quick glance at the icon for the gesture may be sufficient for the user. An animation is the second level: it requires more time but provides more information and explanation. Our design essentially supports a hierarchy of information where there is a time versus amount of information tradeoff.

A hierarchic view of information can also be applied to the way in which gestures themselves are revealed. For some gestures, it is sufficient just to show a static picture of the gesture. For other gestures an annotated animation is needed before each one can be understood. Besides an animation, some gestures need to show variations. Finally some gestures, like menu marks, are best revealed incrementally. Depending on the characteristics of a gesture, there are different ways of explaining the gesture. This implies our revelation schemes must support these different forms of explanation. Marking menus, crib-sheets, and animations are instances of different forms of explanation. A complete taxonomy of forms of explanation is future research.

While user testing is needed to refine our design, we feel that this design supports the desired type of information flow. Users can interactively obtain information on gestures and this information is intended to interactively teach them how to use these gestures like an expert. No pen-based system that we know of supports this type of paradigm.

Acknowledgments

We thank the members of the Input Research Group at the University of Toronto, especially Gary Hardock for his user testing, and George Fitzmaurice and Beverly Harrison for their comments on thesis drafts of this research. A portion of this work was performed in the Dynamic Graphics Project laboratory at the University of Toronto. We gratefully acknowledge the financial support for the laboratory provided by the Natural Sciences and Engineering Research Council of Canada, Digital Equipment Corporation, and Apple Computer.

References

1. J. M. Carroll, *What's in a name?* Freeman, New York (1985)

2. C. G. Wolf, "Can People Use Gesture Commands?" *ACM SIGCHI Bulletin,* **18,** pp. 73-74, Also, *IBM Research report RC 11867* (1986)

3. J. D. Gould and J. Salaun, "Behavioral Experiments in Handmarks" *Proceedings of the CHI + GI '87 Conference on Human Factors in Computing Systems and Graphics Interface,* ACM, New York, pp.175–181 (1987)

4. W. Buxton, "The "Natural" Language of Interaction: A Perspective on Nonverbal Dialogues" In Laurel, B. (Ed.) *The Art of Human-Computer Interface Design,* Addison Wesley, Reading Massachusetts, pp. 405-416 (1990)

5. D. A. Norman and S. W. Draper, *User centered system design: New perspectives on human-computer interaction,* Erlbaum Associates, Hillsdale, NJ (1986)

6. B. Shneiderman, *Designing the User Interface: Strategies for Effective Human Computer Interaction"* Addison-Wesley, Reading Massachusetts (1987)

7. N. E. Wiseman, H. U. Lemke, and J. O. Hiles, "PIXIE: A New Approach to Graphical Man-machine Communication", *Proceedings of 1969 CAD Conference Southhampton,* IEEE Conference Publication 51, p. 463 (1969)

8. J. Callahan, D. Hopkins, M. Weiser. and B. Shneiderman. "An empirical comparison of pie vs. linear menus" *Proceedings of CHI '88,* pp. 95-100 (1988)

9. G. Kurtenbach, *The Design and Evaluation of Marking Menus,* Ph.D. thesis, University of Toronto, Toronto, Ontario, Canada (1993)

10. E. R. Pederson, K. McCall, T. P. Moran, and F. G. Halasz, "Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings"*Proceedings of the CHI '93 Conference on Human Factors in Computing Systems,* ACM, New York, pp. 391–398 (1993)

11. G. G. Robertson, A. D. Henderson Jr., and S. K. Card, "Buttons as First Class Objects on an X Desktop"*Proceedings of UIST '91Conference,* ACM, New York, pp. 35-44 (1991)

12. H. Lieberman, "An example-based environment for beginning programmers" *AI and Education: Volume One,* Lawler, R. and Yazdani, M., (Ed.), Ablex Publishing, Norwood NJ, pp. 135–152, (1987)

13. R. Baecker and I. Small, "Animation at the Interface" In Laurel, B. (Ed.) *The Art of Human-Computer Interface Design,* Addison Wesley, Reading Massachusetts, pp. 251–267 (1990)

14. R. E. Cullingford, M. W. Krueger, M. Selfridge, and M. A. Bienkowski, "Automated explanations as a component of a computer-aided design system" *IEEE Transactions on System, Man and Cybernetics,* March/April, pp. 168–181 (1982)

15. P. Sukaviriya and J. D. Foley, "Coupling a UI framework with automatic generation of context-sensitive animated help"*Proceedings of the ACM Symposium on User Interface Software and Technology '88,* ACM, New York pp. 152–166 (1990)