use the term *selection termination* to indicate the action that ends the entire menu selection process. In non-hierarchic case, selection confirmation and selection termination are combined in the same action.

There are many different types of input events that could be used to signal selection confirmation:

• *Pen-up/pen-down*

• *Item entry:* Item entry means selection confirmation occurs the moment the pen enters an item.

• *Boundary crossing:* Boundary crossing means that selection confirmation occurs when the pen crosses the outside border of a menu item.

• *Dwelling:* Dwelling is the act of keeping the pen pressed and not moving for a fraction of a second. A user can avoid issuing dwelling events by keeping the pen moving. Press-and-wait is an example of a dwelling event. However, we distinguish between these two events because press-and-wait signals the entry into menu mode while dwelling signals selection confirmation.

• *Events distinct from pen movement:* This includes things like a button press or an increase in pressure with a pressure sensing pen.

The type of selection confirmation event used affects other design features:

• *mimicking drawing a mark:* Since selection from a hierarchy of menu items involves a series of selection confirmations and we wish to mimic that act of making a mark, an event for selection confirmation that does not interrupt dragging must be used.

• *reselection:* In some cases, a user may desire to change the previewed selection. For example, a user may accidentally move into the wrong item then want to move to the correct item. We refer to this process as reselection. Most menu systems support reselection.

• *pairing command and parameters:* The command compass allows dragging to continue after the final selection confirmation. Dragging is then used to indicate additional parameters to the menu command just selected.

48

Figure 2.5 shows which selection confirmation methods support these features. Item entry is not feasible because it does not allow reselection. Boundary crossing, dwelling and events distinct from pen movements support both reselection and pairing. We discount "events distinct from pen movement" because it requires additional input sensors like pen buttons or a pressure sensing pen.

Figure 2.5 indicates that boundary crossing and dwelling are the only applicable choices. Boundary crossing is preferable because a visible boundary (i.e., the edge of a menu) gives precise information as to when selection will occur. This information is not visible if dwelling is used. Furthermore, waiting for a dwelling to occur slows interaction. It is also possible to use pen release as a confirmation method if pairing is not required and the item being selected is the last in a series of selections.

We implemented boundary crossing by having selection confirmation occur when the user crossed over the outer edge of a menu item. Specifically, selection previewing occurred as long as the user stayed within the circle of the menu. Selection confirmation occurred when the user moved outside the circle. We discovered, in practice, that boundary crossing created a problem. As a user moves away from the center of the menu to confirm an item, the item's sub-menu pops up when the outer boundary is crossed. Unless a user moves very slowly, one is still moving when the sub-menu appears. This results in one of the items in the sub-menu being selected immediately. If the user is moving fast, the boundary point for the sub-menu may have already been crossed and this results in an erroneous selection confirmation. Even if the boundary point was not crossed, this overshooting in the sub-menu causes reselection to be the first action to occur each time a sub-menu is popped up. This means that users are not rehearsing the movement of drawing a mark, but are rather making a movement which involves reselection. This approach was therefore unacceptable.

To solve this problem, we used a hybrid approach which combines boundary crossing and dwelling. The approach works as follows. As long as the pointer is within some distance from the center of menu, a dwelling event is ignored. Selection preview and reselection are therefore possible without the threat of an accidental dwelling occurring. Once the boundary is crossed, selection preview and reselection are still possible but, if the user dwells, the selected item is confirmed and its sub-menu appears. This allowed users to use coarser movements to make selections without fear of overshooting and selecting from sub-menus.

Dwelling is also consistent with press-and-wait. In both these activities, keeping the pen pressed against the display and holding it triggers the display of a menu.

A selection can also be confirmed without dwelling by releasing the pen at any point in the hierarchy of a menu. This allows any item in the hierarchy to be selected and also signals selection termination.

2.5.6. Mark ambiguities

The current design presents a dilemma if we consider using marks to make selections from hierarchies of menus. The idea behind using marks for selection is

Selection confirmation event	allows mimicking marking?	allows reselection?	allows pairing?
pen release	no*	yes	no
item entry	yes	no	yes
boundary crossing	yes	yes	yes
dwelling	yes	yes	yes
events distinct from pen movement	yes	yes	yes

(* yes in the non-hierarchic case)

(as long as the pointer is kept moving)

Figure 2.5: Different selection confirmation methods characteristics.

50

that selection will be fast and fluid. This implies that we do not desire or expect a user to "include" dwellings when making selections using marks. This would be unnatural and slow the marking process.

A problem can occur if dwellings are not included when making marks. Consider a selection from a hierarchy that is two levels deep. Suppose the user makes a straight line mark. Does the mark correspond to a selection from the parent menu or the child menu? Figure 2.6 shows the problem. If dwellings no longer occur we cannot disambiguate the selection. If we base the interpretation on boundary crossing, then the mark is unambiguous. Unfortunately, this makes the size of a mark affect its interpretation (i.e., the marks cannot be scaled).

One solution to this problem is called *no category selections.* It is based on the observation that items which have subitems are generally categories of commands, not commands themselves, and selecting a category is not a meaningful operation. For example, when using linear hierarchic menus on the Macintosh, selecting the "font" category leads to a menu of commands that change the font. Selecting "font" by itself (i.e., releasing the mouse button when "font" is selected) performs no operation. Therefore we assume that there is no need to select a category. Thus, we can consider any straight line to be a selection into a submenu (case (b) in Figure 2.6). Note that this permits selection of certain menu items that are embedded in submenus by drawing a short straight mark. We recommend designers put the most popular item in a category in this position to promote efficiency.
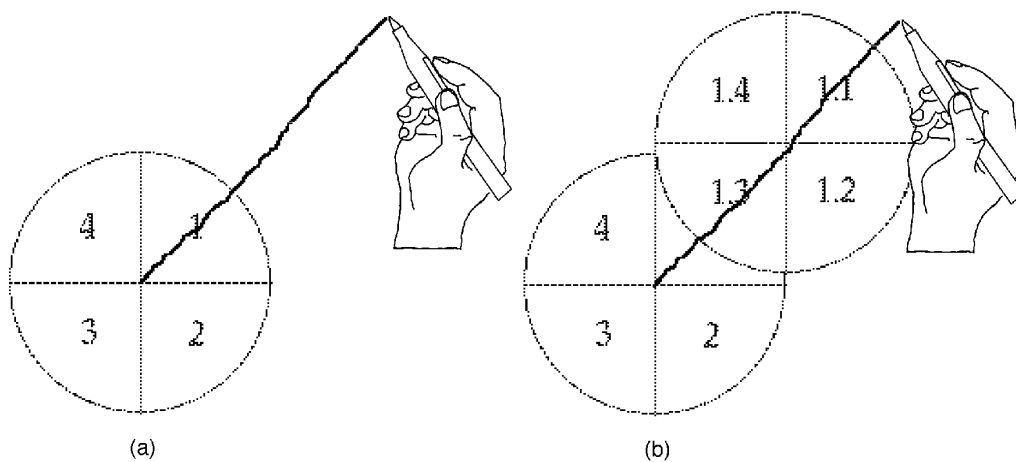
Figure 2.6: Ambiguity in selecting from a hierarchy of menu items two levels deep using a mark. Overlaid grayed menu show possible interpretations. In (a), the interpretation is the selection of item 1. However, (b) is another interpretation according to boundary crossing rules (the selection of item 1.1). Interpretation by boundary crossing is sensitive to the size of marks.

No category selections breaks down when the depth of the hierarchy is greater than two. Suppose a user makes a "^" mark as shown in Figure 2.7 (a). The start of the mark and the change in direction within the mark indicate two points of menu selection. However, what indicates selection from the third level of menu? Figure 2.7 shows this problem. Once again, boundary crossing can be applied to derive an unambiguous set of menu selections but this results in unscalable marks.

There are several solutions to this problem which preserve scaling. The first solution, referred to as the *no-oping* (from the phrase "no operation"), is to simply not permit a series of menu selections that result in a straight line. One way of doing this involves making the item in the child menu that "lines up" with the selection angle of the parent a null operation. This ensures that the beginning of a selection of a non-null item from a child menu is indicated by a change in angle. Unfortunately, this "wastes" a useful sector in a menu.
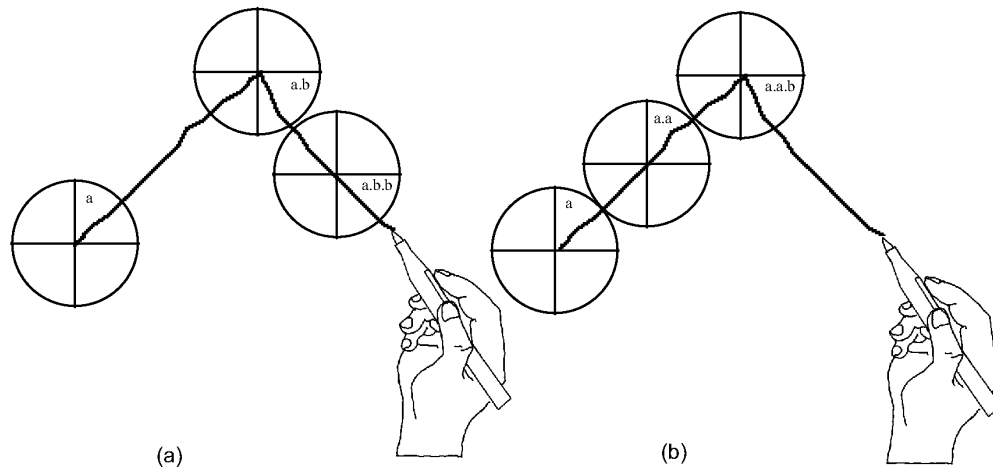
52

Figure 2.7: Possible interpretations of mark when selecting from hierarchies greater that two levels deep. The straight line sections of the mark have no artifacts to indicate whether the selection at that point is being made from the parent or from the child.

A second solution is *axis-shifting*. This involves rotating child menus such that no item appears at the same angle as an item in the parent menu. Figure 2.8 shows an example of this technique. Axis-shifting involves aligning the boundary between two items in the child menu with the selection angle of the parent item. This ensures that the beginning of a selection from child menu is indicated by a change in angle. Axis-shifting avoids the wasted sectors that occur with no-oping.

This discussion has presented four solutions to hierarchic menu design which are intended to produce an unambiguous vocabulary of marks. The four solutions are: boundary crossing, no category selections, no-oping, and axis-shifting. The aspects of the design that are affected by these solutions are: the ability to select any item within the hierarchy, the ability to have mark interpretation independent of the size of a mark, the ability to select leaf items with a single straight line, and the ability to have all items in a menu active. These aspects may also vary relative to the depth of the menu. Figure 2.9 summarizes this design space.

A solution can be chosen based on the demands of the menu. If menus are only one or two levels deep and menu categories do not need to be selected, then no category selections will work. Boundary crossing and axis-shifting are suitable when hierarchies are more than two levels deep and category menu items need to be

53

selected. Boundary crossing is also an acceptable solution if category items need to be selected and mark scaling is not an issue.
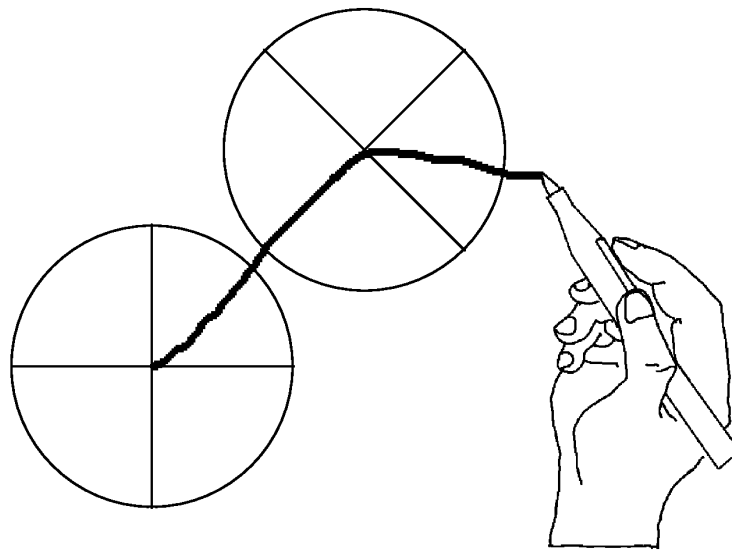

Figure 2.8: Axis shifting rotates a child menu such that child menu items do not appear on the same angle as the parent menu item. This results in a mark language where selection confirmations are indicated by changes in angle. With this scheme marks can be drawn at any size.

Policy	no depth limit?	select any item?	marks scalable?	allows "straight lining"	all items active?
boundary crossing	Yes	Yes	No	Yes	Yes
no-oping	Yes	Yes	Yes	No	No
no category selections	No (2)	No (except in 1 deep case)	Yes	Yes	Yes
axis-shifting	Yes	Yes	Yes	No	Yes

Figure 2.9: Policies that avoid ambiguous interpretation of marking menu marks.

2.5.7.   Display methods

There are several design options which concern how menus are displayed:

• *Menu trail* refers to leaving parent menus displayed as a user descends a hierarchy of menu items.

• *Menu overlap* refers to displaying child menus over the top of parent menus.

These methods become important when backing up in a hierarchy of menus.

2.5.8.   Backing-up the hierarchy

The ability to back-up in a hierarchy of menus is useful for browsing menu items and correcting mistakes. Backing-up can be one of three types: back-up only to the parent menu, back-up to any ancestor menu, back-up to any ancestor menu item. Backing-up can be accomplished in several ways. Pointing to an item can trigger a back-up to the item, or an explicit action can trigger a back-up (i.e., tapping the pen triggers a back-up to the parent menu). A combination of these two methods can be used (i.e., tapping on an item to back-up to it). Lifting the pen is already used to indicate selection termination, so the back-up technique is restricted to pointing while the pen is being dragged.

55

Backing-up brings the roles of menu trail and menu overlap into play. Pointing to the item in order to back-up to it requires that item be displayed on the screen. Therefore a menu trail must be provided. However, child menu items may cover up parent items making it impossible to point to "covered" items. The design must ensure that parent items are not covered up.

Design requirements dictate that backing-up in marking menus operates like backing-up in traditional drag-through hierarchical menus: to back-up to a parent menu item, a user points to it; the system then closes the currently displayed child menu and displays the child menu of the parent item. We can adopt this scheme for marking menus but it reduces the advantage of radial menu selection. Figure 2.10 shows the problem that occurs. A selection from a child menu may result in pointing to a parent menu item and this causes an unintended back-up. A prototype implementation of marking menus revealed this to be a real problem. The problem could be avoided if a user is "careful", but this tends to slow users down.
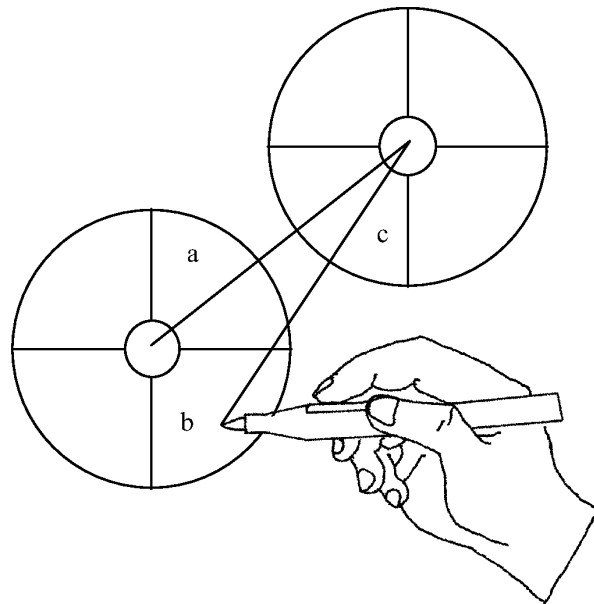
Figure 2.10: A problem with the backing up by pointing to a parent item. Is the user selecting item a.c or backing up to item b?

To solve this problem, we could restrict marking menus to operate like linear menus where selection occurs only if the user is pointing inside a menu item. This has two

major disadvantages. First, it selection sensitive to the length of strokes, and second, it massively reduces item size from a sector of the entire screen to the small sector of the menu.

The solution is to reduce the size of the back-up targets. This is done by restricting the back-up targets to the center hole of the parent menus. This drastically reduces the probability of accidentally pointing to a back-up target. Furthermore, we constrain the user to dwell on a center before back-up takes place. This allows the user to "pass through" centers without backup occurring. Figure 2.11 shows this back-up scheme.

This approach has the restriction of only allowing back-up to parent menus. Backing up to a parent menu and displaying another one of the child menus cannot be combined in the same operation. Some hierarchic linear menus allow this. However, this restriction permits fast and unconstrained selection when moving forward in the hierarchy, while still allowing back-up.

This back-up scheme has several more advantages. First, one can back-up to any parent menu, grandparent menu, etc. Second, menu overlap can occur just as long as menu centers do not get covered. Finally, because backing-up actually returns the cursor to parent menus, rather than redisplaying parent menu at the cursor location, this reduces the chances of menus "walking off" the screen (this problem is further discussed in Section 6.2.3).
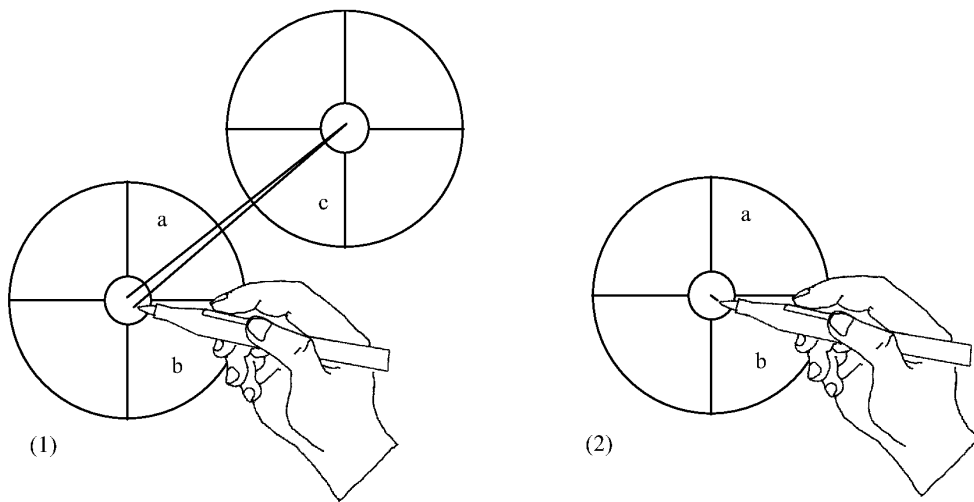
Figure 2.11: Backing-up in hierarchic marking menus. In (1) the user moves into the center of a parent menu and dwells momentarily. In (2) the system senses the dwelling and backs-up to the parent menu by removing the child of item a. Selection may then continue from the parent.

2.5.9.   Aborting selection

Most menu systems have a way of specifying a null selection. Generally this is accomplished by selecting outside a menu item. As explained previously, marking menus allow selection to occur outside the item to make selection easier. To circumvent this problem, the center hole of a menu is used to indicate no selection. Lifting the pen within the center hole results in the menu selection being aborted.

A mark may be also be aborted. This involves either lifting the pen before the mark is complete or turning the mark into an uninterpretable scrawl while drawing it.

2.5.10.   Graphic designs and layout

During everyday use of marking menus we observed some problems with a "pie" graphical representation. First, as the number of items in the menu increases and the length of labels increases, the size of the pie grows rapidly. This creates several problems. First, having large areas of the screen display and undisplay is visually annoying. Second, a large menu occludes too much of the screen. In many situations, a menu associated with a graphical object must be popped up over the

58

object. The problem is that displaying the menu completely hides the object. This results in the context of the selection being lost. Third, large menus take time to display and undisplay. In most systems, the image "underneath " a menu is saved before a menu is displayed, and restored when a menu is undisplayed. When a menu is very large, these operations take considerable amounts of time because large sections of memory are being copied to and from the display. Also, algorithms for sizing and laying out labels within the pie of the menu can be quite complex. This makes the implementation of menu layout procedures complex. Complex computations may also delay the display of menus.

To solve these problems we designed an alternate graphic layout for marking menus called "label"[10]. Figure 2.12 shows an example. This alternate design has several advantages over a pie representation. First, it reduces the amount of screen that changes when a marking menu is displayed and undisplayed, and therefore, it reduces visual annoyance. Second, it occludes less of the screen than a pie representation because only the menu center and labels are opaque. Thus more of the context underneath a menu can be seen. This design also reduces the amount of memory that must be copied to and from the display, and hence it reduces the amount of time needed to display a menu.

Another issue of graphical layout is the problem of displaying menus near an edge or corner of the screen. Pie menu systems deal with this issue by using a technique called "cursor warping". Unfortunately, cursor warping is not suitable for pen-based systems. In Chapter 6, we further discuss this issue and describe an alternative to cursor warping.

Although not shown in Figure 2.12, marking menus have many standard features found in traditional menus. For example, marking menus allow grayed-out and checked items. Also, if an item has a submenu, a small circle appears to the right of the label. The intention is that this circle represents the center hole of the submenu. We also found it valuable to hide the labels of parent menus, thus reducing screen clutter. The only portion of a parent menu that is displayed is the center hole (so a user can point to it to back-up). We have also experimented with transparent menus

---

[10] We acknowledge Mark Tapia for his assistance in designing and implementing the alternate graphical layout for marking menus
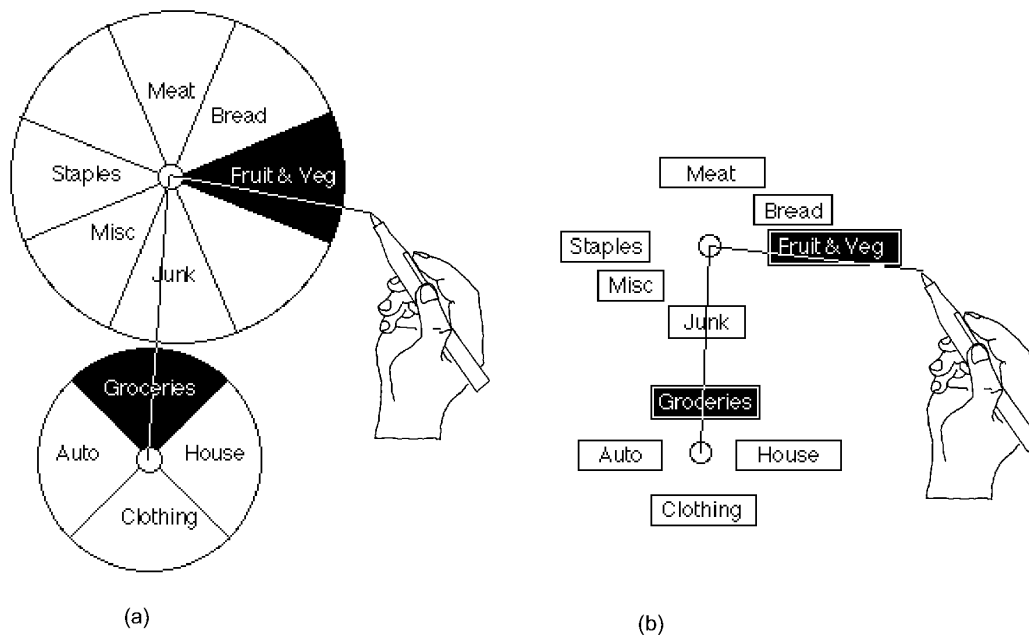
Figure 2.12: An alternate graphic representation for a radial menu "label". Rather than displaying "pie" shapes (a), only the labels and center are displayed (b). The menu then occludes less of display and can be displayed faster.

and graying out parent menus but a full discussion of these experiments is beyond the scope of this dissertation.

2.5.11. Summary of design

The previous sections described and discussed various design features and options of marking menus. We now summarize the features and indicate which design options we elected to use.

Marking menus use discrimination by angle. Selection previewing in menu mode is supported by dragging the pen into an item, and the item being highlighted. Selection confirmation is indicated by a combination of boundary crossing and dwelling. Selection termination is indicated by pen up.

To avoid mark ambiguities, we recommend three possible strategies: no-oping, no category selections and axis-shifting. If menus require only a few items, no-oping may be a suitable solution. If menus are only two levels deep and category selection is not required, no category selection is a suitable solution. If menus require many

menu items, and are more than two levels deep, axis-shifting must be used. In practice, we used no category selection in many situations.

Making a selection in menu mode leaves a menu trail but only the center of parent menu is displayed. We found in practice this reduces the visual clutter the would be caused by the display of inactive parent menu items. Menus are allowed to overlap, but because only the center of parent menu is displayed, this generally does not cause visual confusion.

In menu mode, selection can be aborted by terminating the selection while pointing to the center hole of a menu. In mark mode, selection can be aborted by turning the mark into a "scribble".

If a user dwells while drawing a mark, the system indicates the menu items that would be selected by the mark by displaying the menus "along" the mark. The system then goes into menu mode. This process, called mark confirmation, can be used to verify the items that are about to be selected by a mark or a portion of a mark.

Marking menus can be displayed in either a "pie" representation or a "label" representation. A "label" representation is suitable when there is a need to minimize the amount of screen occluded by the display of the menu.

2.6.   SUMMARY

The success of an interaction technique depends not only on its acceptance by users but also on its acceptance by interface designers and implementors. An "industrial strength" interaction technique must not only be effective for a user, but also have the ability to co-exist with other interaction techniques, other paradigms, and differing features of the software and hardware. Because of these demands, as in many other interaction techniques, our motivation and design behind marking menus is complex. What appears on the surface as a simple interaction technique is actually based on many different motivations and has many design subtleties and details.

In this chapter we defined marking menus and described the various motivations for developing and evaluating them. These included providing marks for functions which have no intuitive mark, supporting unfolding interface paradigms,

61

simplifying mark recognition, maintaining compatibility with existing interfaces, and supporting both novice and expert users. We are also motivated to study marking menus as a way to evaluate the design principles they are based on.

We then outlined the issues involved in evaluating marking menus and proposed an initial design. The major parameters to be evaluated concern the question of how much functionality can be loaded on a marking menu. Essentially our research focus is on establishing the limitations of marking menus so interface designers who are utilizing marking menus can design accordingly. The remaining chapters explore the limitations and characteristics of the design.

Chapter 3: An empirical evaluation of non-hierarchic marking menus

This chapter addresses basic questions about marking menu design variables: how many items can marking menus contain; what kinds of input devices can be used in conjunction with marking menus; how quickly can users learn the associations between items and marks; how much is performance degraded by not using the menu; and whether there is any advantage in using an ink-trail. This chapter describes an experiment which addresses these questions. The approach is to pose specific hypotheses about the relationship between important design variables and performance, and then to test these hypotheses in the context of a controlled experiment. The results of the experiment are then interpreted to provide answers to the basic questions posed above.

In this experiment we limit our investigation to non-hierarchic marking menus. We do this for several reasons. First, this experiment serves as a feasibility test of non-hierarchic marking menus. If non-hierarchic marking menus prove feasible, then an investigation of hierarchic marking menus is warranted. Second, we feel that the characteristics of non-hierarchic marking menus must be understood before we can begin to investigate hierarchic marking menus. Our findings on non-hierarchic marking menus can then be used to refine our design and evaluation of hierarchic marking menus. Third, this experiment addresses many factors. To include the additional factor of hierarchic structuring would make the experiment too large and impractical.

To date there is little research applicable to our investigation. Callahan, Hopkins, Weiser, and Shneiderman (1988) investigated target seek time and error rates for 8-item pie menus, but concentrated on comparing them to linear menus. In particular

63

they were interested in what kind of information is best represented in pie menu format. Section 2.3.1 described their results.

Our experiment focuses on selecting from marking menus using marks. To address the questions posed at the start of this chapter, the experiment examines the effect that the number of items in a menu, choice of input device, amount of practice, and presence or absence of an ink-trail or menu, has on response time and error rate.

3.1. THE EXPERIMENT

3.1.1. Design

In this experiment, we varied the number of items per menu and input device for three groups of subjects and asked them to select target items as quickly as possible from a series of simple pie menus. One group selected target items from fully visible or "exposed" menus (Exposed group). Since there is little cognitive load involved in finding the target item from menus which are always present, we felt that this group would reveal differences in articulation performance due to input device and number of items in a menu.

Two other groups selected items from menus which were not visible ("hidden" menus). In one group, the cursor left an ink-trail during selection (Marking group), and in the other, it did not (Hidden group). The two hidden menu groups were intended to uncover cognitive aspects of performance. Hiding the menus would require the added cognitive load of either remembering the location of the target item by remembering or mentally constructing the menu, or by remembering the association between marks and the commands they invoke through repeated practice. Comparing use of an ink-trail with no ink-trail was intended to reveal the extent to which supporting the metaphor of marking and providing additional visual feedback affects performance. The Exposed group provided a baseline to measure the amount that performance degraded when selecting from hidden menus.

3.1.2. Hypotheses

We formed the following specific hypotheses to address the questions posed at the start of this chapter:

64

How much is performance degraded by not using the menu?

Hypothesis 1. Exposed menus will yield faster response times and lower error rates than the two hidden menu groups. However, performance for the two hidden groups will be similar to the Exposed group when the number of items per menu is small. When the number of items is large, there will be greater differences in performance for hidden versus exposed menus. This prediction is based on the assumption that the association between marks and items is acquired quickly when there are very few items. As the number of menu items increases, the association between marks and items takes longer to acquire, and mentally reconstructing menus in order to infer the correct mark becomes more difficult.

How many items can marking menus contain?

Hypothesis 2. For exposed menus, response time and number of errors will monotonically increase as the number of items per menu increases. This is because we assume that performance on exposed menus is mainly limited by the ease of articulation of menu selection, as opposed to ease of remembering or inferring the menu layout. We know that performance time and errors monotonically increase as target size decreases, all else being equal (Fitts, 1954).

Hypothesis 3. For hidden menus (Marking and Hidden groups), response time will not solely be a function of number of items per menu. Instead, menu layouts that are easily inferred or that are familiar will tend to facilitate the cognitive processes involved. We predict that menus containing eight items can be more easily mentally represented than those containing seven items, for example. Similarly, a menu containing twelve items is familiar since it is similar to a clock face, and thus we predict it is more easily mentally represented than a menu containing eleven items.

What kinds of input devices can be used in conjunction with marking menus?

Hypothesis 4. The stylus will outperform the mouse both in terms of response time and errors. The mouse will outperform the trackball. This prediction is based on previous work (Mackenzie, Sellen, & Buxton, 1991) comparing these devices in a Fitts' law task (i.e., a task involving fast, repeated movement between two targets in one dimension).

Hypothesis 5. Device differences will not interact with hidden or exposed menus, or the presence or absence of marks. Differences in performance due to device will

65

not depend on whether the menus are hidden or exposed, or whether or not marks are used. The rationale for this is that we assume performance differences stemming from different devices are mostly a function of articulation rather than cognition. We also assume that the articulatory requirements of the task are relatively constant across groups.

Is there any advantage in using an ink-trail?

Hypothesis 6. Users will make straighter strokes in the Marking group. We based this prediction on the assumption that visual feedback is provided in the Marking group and also that hidden menus support the "marking" metaphor as opposed to the "menu selection" metaphor.

How quickly can users learn the associations between items and marks?

Hypothesis 7. Performance on hidden menus (Marking and Hidden groups) will improve steadily across trials. Performance with exposed menus will remain fairly constant across trials. This prediction is based on belief that articulation of selection (or simply executing the response) will not dramatically increase with practice since it is a very simple action. Performance on hidden menus, however, involves the additional cognitive process of recalling the location of menu items. We believe this process will be subject to more dramatic learning effects over time.

3.1.3.   Method

Subjects. Thirty-six right-handed subjects were randomly assigned to one of three groups (Exposed, Hidden, and Marking groups). All but one had considerable experience using a mouse. Only one subject had experience using a trackball. None of the subjects had experience with a stylus.

Equipment. The task was performed on a Macintosh IIX computer. The standard Macintosh mouse was used and set to the smallest C:D ratio. The trackball used was a Kensington *TurboMouse*, also set to the smallest C:D ratio. The stylus was a Wacom tablet and pressure-sensitive stylus (an absolute device). The C:D ratio used was approximately one-to-one.

Task. Subjects used each of three input devices to select target "slices" from a series of pie menus as quickly and as accurately as possible. The pies contained either 4, 5, 7, 8, 11, or 12 slices. All pie menus contained numbered segments, always beginning

66

with a "1" immediately adjacent and to the right of the top segment. The other slices were labeled in clockwise order with the maximum number at the top (see Figure 3.1 (a)). The diameter of all pie menus was 6.5 cm., and Geneva 14 point bold font was used to label the slices.
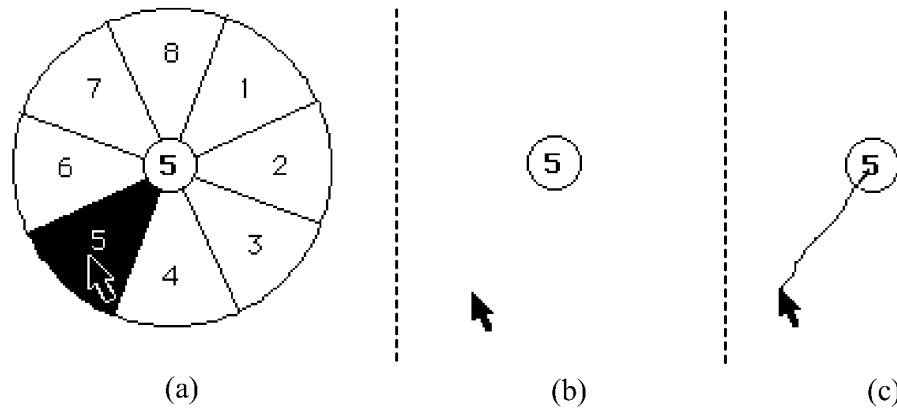

Figure 3.1: Selecting item 5 from an eight-item pie menu (a) in the Exposed group, (b) in the Hidden group, and (c) in the Marking group.

In designing this experiment, a great deal of time was spent discussing what kind of items should be displayed in the pie menus. Menus in real computer applications usually contain meaningful items, but the order in which they appear is not easily inferred. The numbered menus we used, on the other hand, used ordered, meaningless labels. We wanted to approximate the case of an expert user who is familiar with the menu layout. We decided to reduce as much as possible the learning time associated with memorizing the items. Our focus was on the articulation of actions, and the cognitive processes involved in mentally representing or mentally constructing menu layout. Since Callahan et al. (1988) have shown that performance varies depending on the kinds of items represented, using the same kind of items for all menus (numbered items) was an attempt to eliminate this effect. Thus our comparisons between menus with different numbers of items would be more accurate. We acknowledge that both the choice of menu items and their mapping within a menu may have a significant effect on performance. These factors are outside the scope of this investigation.

67

In the Exposed menu group, the entire menu was presented on each trial (Figure 3.1 (a)). The target number corresponding to the slice to be selected was presented when the subject located the cursor within the center circle of the pie menu and either pressed down and held the mouse or trackball button, or pressed down and maintained pressure on the stylus. The subject's task was then to maintain pressure and move in the direction of the target slice. Menu slices would highlight as the cursor moved over them, indicating to the subject a potential selection. A slice would remain highlighted even if the cursor went outside the outer perimeter of the pie. Releasing the button, or pressure, signaled to the system that the highlighted slice was selected. After the selection was made, the menu would "gray out" displaying the menu with the slice selected for a period of 1 second. If an incorrect slice was selected, the Macintosh would beep on release. This marked the end of a trial.

In the Hidden menu group, the task was essentially the same, except that during selection, only the central circle of the pie menu would be visible (Figure 3.1 (b)). After confirming the selection, subjects would receive the same grayed-out feedback as in the Exposed group, indicating which response had been made, and whether or not it had been correct. The Marking group was almost identical to the Hidden group, except that the movement of the cursor with the button depressed left an ink-trail (Figure 3.1 (c)) .

After each trial, subjects received a running score, presented in the lower right-hand corner of the screen. A minimum of 10 points could be obtained for each correct response, with more points scored as response time became shorter. However, subjects were penalized 20 points for errors.[11] At the end of each block of trials, each subject's current performance was shown in relation to the best score obtained by other subjects in the same conditions. The scoring criterion was the same for all groups.

Design and Procedure. One third (twelve) of the subjects were randomly assigned to the Exposed group, one third to the Hidden group, and one third to the Marking

---

[11] This scoring scheme was arrived at by experimenting with different scoring schemes on pilot subjects. We found that the choosen scheme emphasized both accuracy and speed. On average, subject scores were positive and they found this encouraging and fair.

group. Every subject used each of the three input devices (mouse, trackball and stylus). Trials were blocked by device and order of device was counterbalanced.

For each device, *all groups* began by practicing on exposed menus for a total of six trials for each of six different menus, containing either 4, 5, 7, 8, 11 or 12 items. During practice, number of items per menu was blocked and presented in random order. This practice period was intended to acquaint subjects with the feel of the particular input device they were about to use. It also provided an opportunity for subjects to familiarize themselves with the layout of the menus before beginning the timed trials.

Subjects in the Exposed group then moved on to the timed trials, while subjects in the Hidden and Marking groups received a further set of practice trials designed to acquaint them with the "feel" of hidden menus. For this practice session, menus containing both three and six items were used (six trials each) since 3-item or 6-item menus were never used in the actual timed trials. This was a deliberate attempt to equalize exposure to the menus of interest in the three groups.

For the timed portion of the experiment, trials were again blocked by number of items (4, 5, 7, 8, 11, or 12). The order in which the number of items appeared was randomly permuted for each subject. Each subject began a particular block by first studying the menu layout for 6 seconds. They then received a total of 40 trials for each different menu with a short break at intervals of ten trials. Targets were drawn randomly from a uniform distribution with replacement, with the added constraint that no target could be repeated on consecutive trials.

In summary, each subject performed 40 trials on each of the six menus (menus consisting of 4, 5, 7, 8, 11, and 12 items) and using all three devices, resulting in a total of 720 scores per subject. Each group consisted of twelve subjects which resulted in 8640 scores per group. The three different groups provided a total of 25920 scores for the experiment.

3.2. RESULTS AND DISCUSSION

The main dependent variables of interest were response time and number of errors. Response time was defined as the total time from presentation of the target number

69

to confirmation of the selection for error-free trials. An error was defined as an incorrect selection. The means for each group are shown in Figure 3.2.

3.2.1. Effects due to number of items per menu

As expected, increasing the number of items per menu significantly increased both response time ($F(5,55) = 388.4$, $p < .001$) and errors ($F(5,55) = 382.8$, $p < .001$).[12] There were overall performance differences among the groups in terms of errors ($F(2,22) = 21.97$, $p < .001$) but not in terms of response time. However, these main effects are not particularly meaningful because differences among groups depend on the number of items per menu (see Figure 3.3). That is, there was a significant interaction between group and number of items per menu both in terms of response time ($F(10,110) = 3.5$, $p < .001$) and errors ($F(10,110) = 64.7$, $p < .001$).

These results address the first three hypotheses:

(1) As predicted by Hypothesis 1, mean response time was consistently lower in the Exposed group versus the Hidden and Marking groups as the number of items increased. This is supported by the significant interaction between group and number of items per menu (reported above), and by specific comparison tests. No difference was found between the two hidden groups and the Exposed group for menus containing four items. However, for menus containing five items, response times were significantly slower for hidden menus compared to Exposed ($F(1,110) = 6.5$, $p < .001$). The two hidden groups were no different from each other in terms of errors (post hoc comparison of error means, Tukey HSD, $\alpha = .05$), but both produced significantly more errors than the Exposed menu group.

(2) Our second hypothesis predicted that in the Exposed group, response time and errors would monotonically increase as a function of number of items per menu. In the case of errors, this relationship seems to hold. However, this must be qualified by the fact that errors were infrequent and thus floor effects may obscure the true shape of the function.

---

[12] Throughout this disseration we use the F-statistic to evaluate the equality of population means. See Appendix A for an explanation.

Group	Mean RT in sec. (SD)	Mean Number of Errors in 40 Trials (SD)	Mean Percentage Errors
Exposed	0.98 (0.23)	0.64 (1.00)	1.6%
Hidden	1.10 (0.31)	3.27 (3.57)	8.2%
Marking	1.10 (0.31)	3.76 (3.67)	9.4%

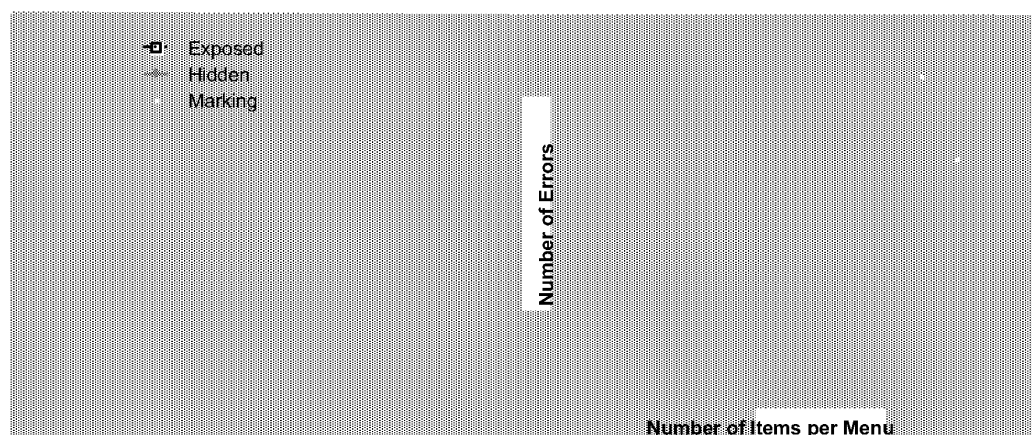*Figure 3.2: Mean response time and number of errors for each experimental group.*

Figure 3.3: Response time and average number of errors (of a total of 40 trials) as a function of number of items per menu and group.

Response time also increased monotonically except for menus containing twelve items. Specific comparisons at the .05 level confirm significant increases in response time from four to five items per menu ($F_{(1,55)} = 16.8$, $p < .001$), and from seven to eight items per menu ($F_{(1,55)} = 7.4$, $p < .01$), but no differences between eleven and twelve items per menu. One possibility is that familiarity with the "clock face" layout may have reduced the time for visual search, thereby reducing overall response time. Another possibility is that this could be a case of diminishing effects. Adding an extra item to a menu containing four items represents a 20% increase in

71

the number of items, whereas, adding an extra menu item to one which contains eleven represents only an 8% increase in number of items.

(3) The pattern of results predicted by Hypothesis 3 is also supported: when menus were hidden, some kinds of menus were easier to evoke or reconstruct from memory than others. This was not purely a function of number of items per menu. The characteristic curve that emerges (Figure 3.3) shows that performance in general does tend to degrade as the number of items per menu increases, but that certain numbers of items do not follow this pattern (i.e., eight and twelve items).

This hypothesis is also confirmed by a series of specific comparisons showing no differences in either hidden menu group for seven versus eight items per menu. Further, performance on menus of twelve items was faster than on menus of eleven items for the Hidden group ($F(1,55) = 11.25$, $p < .001$) and was more accurate than on menus of eleven items in both groups (Hidden, $F(1,55) = 50.96$, $p < .001$; Marking $F(1,55) = 13.51$, $p < .001$). By contrast, for both groups, tests show menus of four items yielded faster response times than menus of five items (Hidden, $F(1,55) = 4.05$, $p < .05$; Marking $F(1,55) = 9.00$, $p < .05$).

The results show that menus containing twelve items in particular may have facilitated performance. Many subjects mentioned that the metaphor of a clock face helped them to select the target item because it could be brought readily to mind. Thus it seems reasonable to suggest that it is the cognitive bottleneck, or the difficulty of evoking the mapping between target and action, that limits performance.

3.2.2.   Device effects

As predicted by Hypothesis 4, subjects performed better with a stylus and a mouse than they did with a trackball. Response time ($F(2,22) = 9.64$, $p < .001$) and errors ($F(2,22) = 11.29$, $p < .001$) were both affected by the type of input device subjects used. Pairwise comparisons (Tukey HSD test, $\alpha = .05$) showed the trackball was both significantly slower and gave rise to more errors than the stylus or mouse. However, contrary to our expectations, there was no difference in mean response time or errors between the stylus and mouse.

Initial analyses supported Hypothesis 5 where we predicted that the effect of input device would not depend on whether or not the menus were exposed, or whether or

72

not there was an ink-trail. Input device did not interact with group, either in terms of response time or errors.[13] However, on closer examination, a more interesting result emerged.

We discovered that in the Marking group, the stylus was significantly faster than both the trackball and mouse with no difference between the trackball and mouse (Figure 3.4). In the Exposed group, the mouse and stylus were faster than the trackball, with no difference between the mouse and stylus. These discoveries were based on separate analyses of variance for each of the three groups on the response time data. There were significant differences among devices in the Exposed ($F(2,22)$ = 10.44, $p < .001$) and Marking groups ($F(2,22)$ = 8.32, $p < .002$), but not in the Hidden group. Tukey tests revealed the superiority of the stylus in the Marking group and the inferiority of the trackball in the Exposed group. No significant interactions between device and number of items were found in any of the three groups. Given these results we cautiously reject Hypothesis 5.
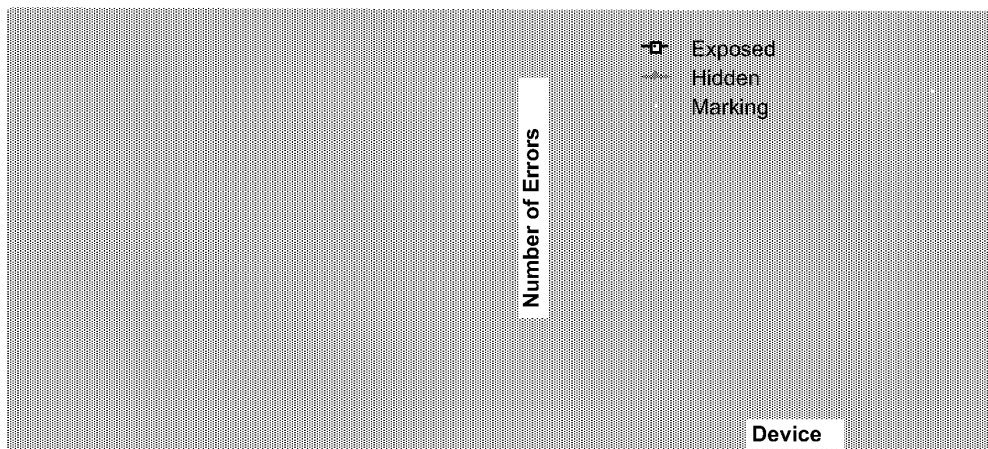

Figure 3.4: Response time and average number of errors (of a total of 40 trials) as a function of device and group.

There may be two reasons for the superiority of the stylus when marks are added to selection from hidden menus. First, it is often difficult to perceive when enough

---

[13] There were also no significant interactions between number of items per menu and device, nor significant three-way interactions (group by number of items by device).

pressure is being applied to the stylus to make a selection. Thus, providing visual feedback when this state is maintained may be important to realize the full potential of this device. Second, providing an ink-trail is consistent with the metaphor of marking with a pen, which may improve performance. Alternatively, failing to support the pen metaphor by not providing the ink trial (Hidden group) may violate users' expectations and thus negatively affect performance.

Separate analyses of the error data within each group further supported the inferiority of the trackball. The trackball was found to be the source of significant device differences in the Exposed $(F(2,22) = 9.92, p < .001)$[14] and Marking groups $(F(2,22) = 9.92, p < .001)$. Pairwise comparisons in the Exposed and Marking groups showed differences between the trackball and the other two devices, and no difference between mouse and stylus.

The finding that the trackball was no more slower or error prone than the mouse and stylus in the Hidden group may be due to the fact that in both the Exposed and Marking groups, visual feedback emphasized the difficulty of articulating the actions of the trackball thereby causing performance to be worse. In the Exposed case, sectors were highlighted as they were selected and it is possible that the trackball caused a great deal of reselection. In the Marking case, users reported that the ink-trail was disturbing in conjunction with the trackball because the paths looked erratic and inaccurate.

3.2.3. Mark analysis

We were interested in seeing if subjects used straight marks when making selections. This was important to discover because, if menu selection tended to be done in some manner other than a straight mark, we could not claim that users rehearse this physical movement when selecting from menus. Thus we would not expect as much transfer of skill between making menu selections and making marks. Another reason we were interested in seeing if subjects used straight marks was related to using marking menus in applications that recognize other marks beside those used in menu selection. Unlike conventional menu selection which is based

---

[14] Both a significant device by menu size interaction $(F(10,110) = 2.47, p < .011)$ and floor effects should make us cautious in interpreting the main effect of device in the Exposed group. However, the fact that the trackball produces consistently more errors on average across menu size, supports the claim that the trackball is outperformed by stylus and mouse.

only on the last location of the cursor, mark recognition systems take the entire shape of the stroke into account. For example, suppose the system also recognizes the symbol "C". A very crooked mark intended to make a selection from a hidden menu might be interpreted as an "C". The success of recognition depends to some extent on knowing the shapes of the strokes that users tend to create. To address these issues we recorded and displayed the path data for users' individual marks. Figure 3.5 shows a typical example.
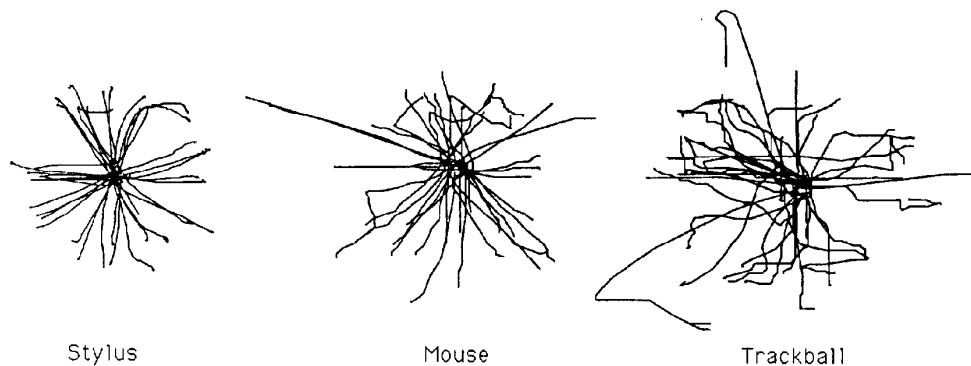

Stylus   Mouse   Trackball

Figure 3.5: The marks a subject used in selecting from a hidden twelve-item menu.

Subjects made approximately straight marks. No alternate strategies such as starting at the top item and then moving to the correct item were observed. However, there was evidence of reselection from time to time, where subjects would begin a straight mark and then change direction in order to select something different.

Surprisingly, we observed reselection even in the Hidden and Marking groups. This was especially unexpected in the Marking group since we felt the idea of drawing a mark does not naturally suggest the possibility of reselection. Hence, we reject Hypothesis 6. It was clear though, that training the subjects in the hidden groups on exposed menus first made this option apparent. Clearly many of the subjects in the Marking group were not thinking of the task as making marks *per se*, but of making selections from menus that they had to imagine. This brings into question our *a priori* assumption that the Marking group was using a marking metaphor, while the Hidden group was using a menu selection metaphor. It may explain why very few behavioral differences were found between the two groups.

75

Reselection in the hidden groups most likely occurred when subjects began a selection in error but detected and corrected the error before confirming the selection. This was even observed in the "easy" four-slice menu, which supports the assumption that many of these reselections are due to detected mental slips as opposed to problems in articulation. There was also evidence of "fine tuning" in the hidden cases, where subjects first moved directly to an approximate area of the screen, and then appeared to adjust between two adjacent sectors.

Strokes produced with the trackball appeared more jagged and less controlled than those made with the mouse or stylus. This is consistent with the statistical results showing that the trackball tends to be slower and less accurate than the stylus or mouse. For four-item menus, most subjects made straighter marks with the stylus than the mouse. The presence or absence of an ink-trail did not appear to make any discernible difference to stroke shape.

3.2.4.   Learning effects

The forty trials for each different menu were divided into eight consecutive blocks. Response time and mean errors were calculated for each five-trial block in order to look more closely at learning effects. Overall, there was a small but steady decrease in response time over trials which was statistically significant ($F(7,77) = 5.79$, $p < .001$). Error rate also showed signs of improving with number of trials ($F(7,77) = 10.52$, $p < .001$).

We have claimed that the major factor limiting performance on exposed menus is the physical accuracy required for the action of selection. The results support this claim. In the case of hidden menus, results support the claim that the factor limiting performance is cognitive. In other words, the time it takes to remember or infer the correct mental representation becomes the overriding factor determining performance. Thus, performance in the Exposed group can serve as a baseline measure that users should approach as they become expert.

Hypothesis 7 states that the cognitive component is the component most affected by learning, as opposed to the articulatory component. Thus, we expect a steady improvement in performance in the two hidden groups, as opposed to fairly constant performance in the Exposed group over time.
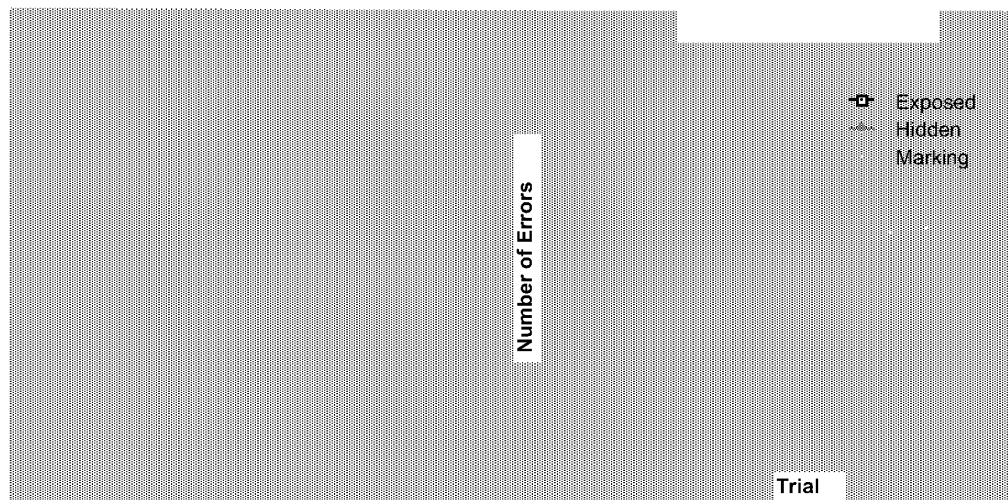
76

Figure 3.6: Group effects in terms of response time and number of errors in five trial intervals.

As is shown in Figure 3.6, response time in the hidden groups appears to improve across trials while the curve for the Exposed group is fairly flat. Errors also remain relatively constant for the Exposed group over trials, while decreasing on average for the two hidden groups. Support for Hypothesis 7 is found in a significant group by trial interaction for response time ($F(14,154) = 2.90$, $p < .001$) and errors ($F(14,154) = 3.15$, $p < .001$).

As a final point, it follows from the above reasoning that we would expect no significant interaction of input device by trial, since type of input device would presumably have the greatest impact on the articulation as opposed to the cognitive component of performance. The fact that no significant interaction of device by trial was found is consistent with expectation.

3.3. CONCLUSIONS

Relative to our seven hypotheses, the results and their implications for design can be summarized as follows:

Hypothesis 1. As predicted, when menus have many items, hiding menus from users both slows their performance and increases their error rate. As number of items per menu increases, added to the problems of articulation is the difficulty of successfully mentally reconstructing the menu layout or remembering the necessary

77

strokes to make menu selections. However, when the number of items is small (only four), there is little or no performance difference, even early in practice.

Design Implications. For ordered sets of commands, users should be as fast and error-free in making marks as in selecting from a visible pie menu of up to four slices. If the commands are not ordered, then it may take more time to acquire the skill. However, command semantics can be exploited. For example, "Open" and "Close" can be positioned opposite to each other, as can "Cut" and Paste". This may speed the learning process and allow users to mark ahead faster. In addition, the most frequently used commands form a very small set, and thus we can be optimistic that these can be invoked successfully with marking menus.

Hypothesis 2 and 3. For exposed menus, the results showed performance declines steadily as the number of items increases. This is probably due to two factors: (1) the increasing reaction time to visually search and choose among alternatives, and (2), the increasing difficulty of articulating the action as targets become smaller.

These results agree with other results concerning the effect of the number of items on performance. Perlman conducted an experiment in which subjects made selections from exposed linear menus (Perlman, 1984). Menus containing 5, 10, 15 and 20 items were used. The menus contained ordered numbers from 1 to 20. Beside each item was a randomly chosen left or right arrow character. The task was to find a target item in the menu and indicate it by pressing the corresponding left or right arrow-key. It was found that the number of items in a menu had a linear effect on the time it takes to find an item. These results agree with our results for exposed menus.

Performance on hidden menus in this experiment was different, however. Instead of a result showing monotonically increasing response times and error rates as a function of number of items, even numbers of items (four, eight, or twelve) appeared to facilitate performance. Not surprisingly perhaps, four-item menus yielded significantly faster and more accurate performance than five-item menus. However, performance on eight-item menus was no worse than performance on menus with one less item. Subjects also reported that the eight-item menu was easy to learn because they could easily mentally subdivide the pie and infer the position of the target slice. Most dramatic was the finding that a twelve-item menu actually yielded faster and more accurate performance than a menu containing only eleven

78

items. We speculate that this difference may be enhanced by familiarity with circles subdivided into twelve sectors, such as in clock faces.

Design Implications. When menus are hidden, overcoming the difficulty of learning and using mental representations of menus can be facilitated by using layouts which exploit known metaphors, or which are easily subdivided. Using an even number of items or laying out items at the points of a compass or hour positions of clock can be used to counteract the increased difficulty of having many items in a menu. The ease with which subjects learned and performed with the twelve-item menu is testimony to the strength of a good metaphor. One could imagine a user remembering a command location or mark by mapping it to an hour/hand position: "undo is at three o'clock".

Hypothesis 4 and 5. The stylus and mouse outperformed the trackball both in terms of response time and errors. Analysis of the paths showed that paths made with the trackball were more jagged and less controlled than those made with the mouse or stylus. The stylus and mouse yielded similar performance, with the exception that the stylus was significantly faster than the mouse when an ink-trail was present.

Design Implications. The results speak strongly against using a trackball for marking menus. Further, subjects' comments suggest that the combination of trackball and ink trial was especially bad. One subject complained of being disturbed by the messy ink-trail left when using a trackball. It seems that the visual feedback provided by the ink-trail only served to emphasize the inadequacy of the paths made by this device.

The performance similarity of the mouse and stylus suggests that either may be appropriate devices for this kind of mechanism. Two cautionary notes should be made, however. First, it is likely that the ink-trail added important feedback to tell the user when the appropriate amount of pressure was being applied to the stylus. This suggests that another kind of stylus (i.e. one with audio or tactile feedback to indicate a "button-click") might have fared better against the mouse in all groups. It also reveals a design deficiency of the stylus that could easily be overcome. Second, while the mouse and stylus yielded similar performance, observation of people using the mouse to make marks other than straight strokes suggests that the mouse may be inferior to the stylus in other situations.

Hypothesis 6. Subjects made essentially straight strokes. However, there was evidence of reselection (where subjects would begin a straight stroke and then change stroke direction in order to select something different) even in the hidden groups. This casts doubt on our initial assumption that subjects in the Marking group would begin to think of the task as making marks, instead of making menu selections. Instead, it suggests that they thought of the task in terms of making selections from the exposed menus they were trained on, which now happened to be hidden. Marks themselves do not afford reselection, whereas pie menus do.

The fact that the marking metaphor was not supported as strongly as we hypothesized may account for the fact that no major differences were found between the Hidden and Marking groups. For example, the presence or absence of an ink-trail did not appear to make any discernible difference to stroke shape.

Design Implications. Since users tended to make straight strokes we are optimistic that users are rehearsing the physical movement required to make marks as they perform menu selection. This bodes well for learning. There was some evidence of non-straight strokes which appeared to be reselection in the Marking group but it was not overwhelming. Perhaps in the context of a mark recognition system a user will learn that reselection results in a mark that cannot be recognized and that reselection is not possible when using a mark.

Hypothesis 7. Performance across trials was uniform for exposed menus but underwent steady and significant improvement across trials for hidden menus (both groups). We argue that the performance limiting factor for exposed menus is the difficulty of articulating selection actions, whereas in the hidden groups the limiting factor is the time it takes to evoke or construct the correct mental representation. Articulation skills were acquired fairly rapidly and reached stable performance. Thus performance in the Exposed group provides a baseline measure that users of hidden menus approach.

Design Implications. The substantial improvement for hidden menus over only 40 trials suggests that if the menus contain meaningful and frequently used commands, users will acquire the necessary skills quickly and easily. Both response time and error rates can be expected to rapidly improve with time. The question of *how much* practice is necessary for hidden menu performance to equal exposed menu performance, and how that varies with number of items per menu is an issue for

80

further research and analysis. Meanwhile, we can be confident that small numbers of items will enable users to quickly begin marking ahead.

3.4. SUMMARY

This chapter investigated basic questions concerning design variables of marking menus: how many items can marking menus contain; what kinds of input devices can be used in conjunction with marking menus; how quickly do users learn the associations between items and marks; how much is performance degraded by not using the menu; is there any advantage in using an ink-trail. An experiment addressed these questions by varying the number of items per menu and input device for three groups of subjects, and asking them to select target items as quickly as possible from a series of simple pie menus. One group selected from menus that were visible at all times, another group selected from menus that were hidden, and the final group selected from menus that were also hidden, but had the additional visual feedback of a cursor ink-trail. The differences in group conditions were intended to separate articulation and cognitive aspects. The experiment compared selection times and error rates. In addition, learning effects were analyzed.

The results of the experiment indicate that non-hierarchic marking menus, or specifically the action of using a mark to select from a menu, is a useful idea. Our results indicate that: (1) four, eight and twelve items menus are suitable for marks; (2) if that number of items is kept low (e.g., four, eight and twelve), users will be able to use marks very early in practice; (3) higher numbers of items are possible but require more practice; (4) for non-hierarchic menus, users will perform as well with the mouse as they would with the stylus/tablet. Using a trackball, however, will be slower and more error-prone than using a mouse or stylus/tablet.

In terms of using marking menus in an application, the results indicate that a designer should attempt to use four, eight or twelve item menus. For example, if seven commands are to be placed in a menu, the designer should use an eight-item menu and leave one item blank or duplicate one of the more popular commands in the extra item. Although this experiment did not address this issue, it may also be also be advantageous to maintain consistent subdivisions for menu items. For example, use four and eight item menus (items on 45 angles) but not twelve item menus (items on 30 angles).

The results are encouraging because there are many applications where menus which have a small number of items could be effective. For example, *Microsoft Word* has seven groups of function icons that appear in the "ribbon" and "ruler" display area. These icons could be grouped into seven marking menus containing four or less items. Each group of icons could be replaced by a single icon which when pressed displays a four-item marking menu. The elimination of icons would allow space to display more text, or other or larger function icons (larger icons make pointing to them easier). The graphics editor in Microsoft Word already has tool pallet icons that work this way but uses pop-up linear menus. The popular Macintosh drawing program called *Canvas* also uses a similar scheme. Many of the menus that pop up from tool pallets icons in Canvas have twelve or fewer items.

While there are many situations where menus with twelve items or less may be sufficient, there are also many situations where menus contain more that twelve items. For example, font menus, large color pallets and paragraph style menus commonly contain more than twelve items. Chapter 5 shows that hierarchic marking menus make it possible to use a mark to select from a large number of items.

Given the results of this experiment, we can now apply them to the design of hierarchic marking menus. We recommend that hierarchic marks contain only menus with even numbers of items and the number of items be less than twelve. Because the poor performance of the trackball in this experiment, it would not be suitable for hierarchic marking menus. Also it would be worthwhile to see if the mouse performs as well as the stylus on "zig-zag" marks. Chapter 5 applies these design recommendations and evaluates hierarchic marking menus.

Despite the value of such controlled studies, there are a number of questions which can only be answered by careful design and implementation of marking menus in real applications. How long will it take for users to start using marks? How intensely will users use marks? What are the issues involved in integrating such a mechanism into a larger, more complex interface? Chapter 4 addresses these types of questions by means of a case study of user behavior using a marking menu for a real task.

82

Chapter 4: A case study of marking menus

The previous chapter has developed an empirical understanding of non-hierarchic marking menus. From this understanding, guidelines for designing marking menus and interfaces that use marking were generated. In this chapter we report on a study which applies those guidelines to the design of marking menus in an application and we evaluate user behavior while operating this application. The application was designed to solve a real world task and was used in accomplishing real work for a project not related to this thesis. The intention was to gain insight on integrating marking menus with other interface components and to find out how well marking menus perform in everyday practical work situations.

4.1. DESCRIPTION OF THE TEST APPLICATION

A conversation analysis/editor program, named *ConEd*, developed at University of Toronto, was used as a test application for marking menus (Sellen, 1992). By digitizing audio from a conversation among four people, data were collected concerning who is speaking and when. The conversation analysis/editor program is then used to display this data in a "piano roll" like representation. The program runs on a Macintosh computer. Figure 4.1 shows a typical display of the data window. The y-axis represents the four participants in the conversation, and the x-axis represents time. A black rectangle indicates that a particular person is speaking for a duration of time (this is referred to as an event). The window can be scrolled to reveal different moments in the conversation. Besides displaying the data, the application can be synchronized to a video recording of the conversation. As the video plays, the application moves a horizontal bar across the window to indicate

83

the current location in the conversation. If the bar moves past the right side of the display, the application automatically scrolls to the next section of conversation.
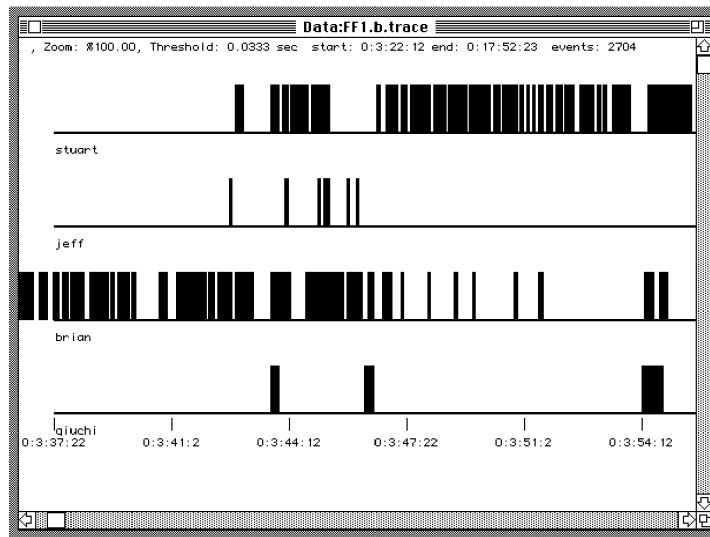

Figure 4.1: The "piano roll" representation of speaker versus time in ConEd.

Data can be edited as well as viewed with this application. Such things as coughs and extraneous noises need to be deleted. Other pieces of conversation, such as laughter, must be tagged for later analysis. Very often events must be added or extended because the automated speaker tracking system was not accurate enough.

Typically, a user sits in front of the Macintosh and video monitor, watching the video and editing events in real-time. Most of the time, a user operated the video transport with the left hand and the mouse with the right hand.

A marking menu triggers the six most frequently used commands, which consisted of commands that coded and edited the blocks of speech. The amount of coding and editing required was extremely high. Over 18 hours of operation, the two users performed 5,237 selections.

4.2. HOW MARKING MENUS WERE USED

4.2.1. The design

84

Figure 4.2 shows the marking menu used in ConEd. This menu can be popped up by pressing-and-waiting with the mouse in the "piano roll" window. Alternatively, a mark can be made to select the command. A user can issue six commands using this menu: laugh, delete, add, fill-in, ignore, and extend.
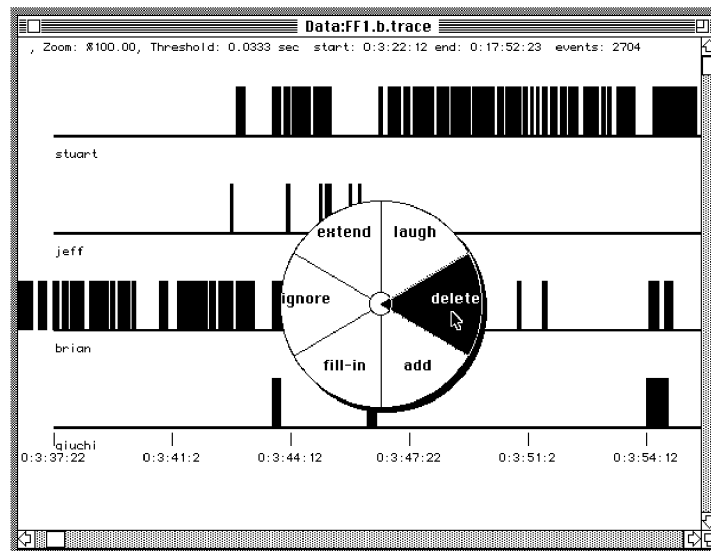

Figure 4.2: The six most frequently used editing commands are placed in a marking menu in ConEd.

Delete: The "delete" command deletes events. If the starting point of the delete selection/mark is made over an event, then that event is deleted. If the starting point is not over an event, then the events lying between the starting and ending points of the selection/mark are deleted. See Figure 4.3.
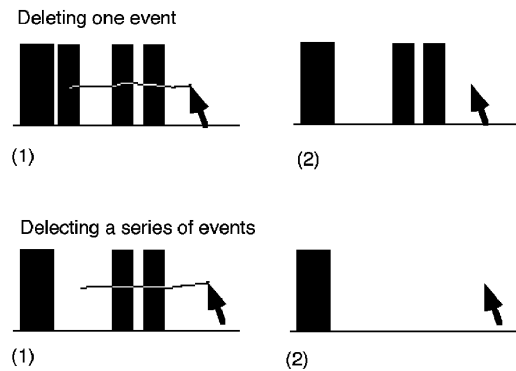
85

Deleting one event

(1)   (2)

Deleting a series of events

(1)   (2)

Figure 4.3: Events can be deleted one at time, by pointing to the event, or in a series by drawing over a series events.

Add: "Add" allows new events to be added. The starting point of the add selection/mark defines the beginning of a new event. The starting point of the following add selection/mark defines the end point of the new event and causes it to be displayed. If add is performed over an existing event, it is disregarded. See Figure 4.4.
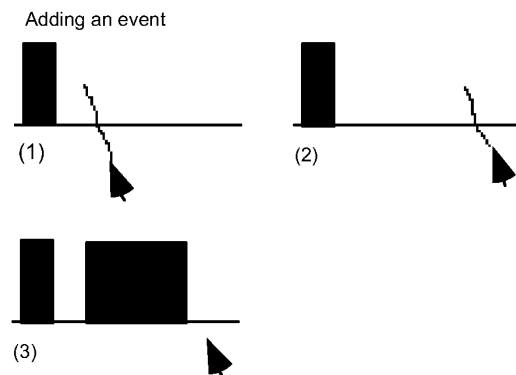
Adding an event

(1)   (2)

(3)

Figure 4.4: Events are added by specifying a starting point followed by and endpoint.

Extend: "Extend" elongates an event. The starting point of the extend selection/mark defines the length of the elongation. Either the start or the end of an event can be extended. If the selection/mark is made between two events, the event whose starting or ending point is closest to the starting point of the selection/mark is elongated. If extend is started over an event, it is ignored. See Figure 4.5.
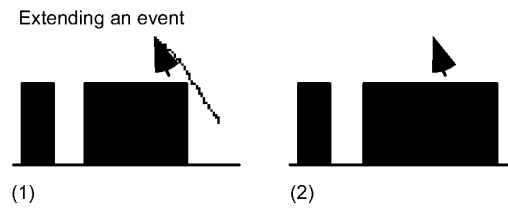
86

Figure 4.5: Events can be extended by pointing to the location of the extension.

Fill-in: "Fill-in" allows a gap of silence between two events to be filled. The two events are replaced by one long event. The starting point of the selection/mark indicates the gap to be filled. If Fill-in is ignored if started over an event. See Figure 4.6.
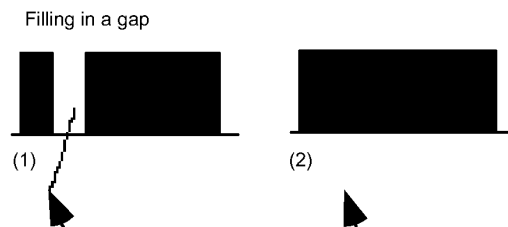

Figure 4.6: Gaps between events can be filled in by pointing to the gaps.

Ignore and Laugh: "Ignore" and "Laugh" allow events to be coded as special types. For example, speaking events generated by laughter must be tagged so they can be excluded from analysis of the conversation. Back-channel events (i.e., someone saying "uh huh" or "yes" but not trying to interrupt while another person is talking) must also be tagged. The starting point of the ignore or laugh selection/mark defines the event being coded. Either command is disregarded if not started over an event. See Figure 4.7 and 4.8.
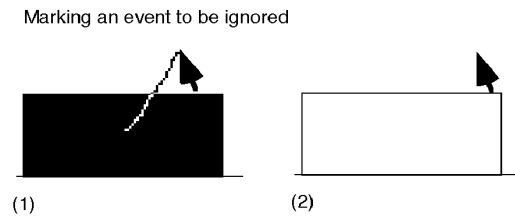
87

Marking an event to be ignored

(1)   (2)

Figure 4.7: An event can be marked to be ignored by pointing to it.

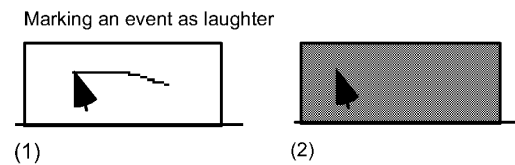Marking an event as laughter

(1)   (2)

Figure 4.8: An event can be marked as laughter by pointing to it.

4.2.2. Discussion of design

Menu item choice

ConEd has more commands than the six contained in the marking menu. There are several reasons for placing this particular set of commands in a marking menu. First, the experiment in Chapter 3 showed that even numbers of items, up to twelve, enhance marking performance. Hence, six is within this range. Second, a requirements analysis told us that these six commands are the most frequently used. This implied several things. First, it would be advantageous if these commands could be invoked quickly. Therefore, marks would be suitable for these commands since marks can be issued very quickly. Second, these commands would be good candidates for marking menus because using the commands frequently would help a user memorize the associations between marks and commands. This, in turn, would lead to users using the marks.

Spatial aspects

Use of end points: While the marks used in marking menus are very simple, other features of a mark besides its angle can be used. The starting and ending points of a mark are obvious candidates. Features of a mark have been used in a similar manner by previous researchers (Coleman, 1969; Rhyne, 1987).

88

A requirements analysis revealed that the most frequent operations would involve selecting an event and applying an operation to it. Thus, marking menus were used in an object oriented manner—the starting point of the selection/mark indicates the object of the command. Note that this is not always the case. For example, the extend command does not point to an event to be extended but to the location of the extension. The particular event to be extended is inferred by the system. However, we found that this inconsistency caused no problems for the user.

The combination of pointing and marking produces the feeling of directness one gets when pointing and moving in objects in direct manipulation interfaces. When using marks in ConEd, there is no sensation of explicitly making a selection before applying an operation.

Use of horizontal/vertical dimension: Spatial commonalties between the representation being edited and the direction of menu items can be used to determine the assignment of directions to commands. For example, horizontal and vertical aspects of the marks can be exploited. Specifically, the direction of a mark means the objects along that direction can be selected using the mark. The delete command is an example of this. Preliminary design testing indicated that deleting a series of horizontal events was a very frequent operation. This meant putting the delete command at a horizontal menu position would allow deletion of several events in a row. This "trick" was found to be very useful.

Spatial commonalties can also be used to provide mnemonics to help recall the mark associated with a command. The add and extend commands are examples of this. Both these commands require a vertical time location value. A common way to indicate location along the horizontal is by a vertical "tick". This serves as a mnemonic for the marks associated with these commands.

Temporal aspects

Time versus space pointing: There are many temporal aspects of a mark that can be used. For example, the speed of drawing (i.e., fast or slow, fast at the start then slow at the end) or the time when a drawing occurred can be used. The aspect we exploited is time-based drawing. Specifically, the add command has two modes of operation. The first mode has been described already—the starting location of the mark is used to define the start or end of the event being added. However, if ConEd is synchronized to the playback of a video tape of the conversation, the start or end

89

point of an event is defined by the current playback location of the video, not by the spatial position of the mark. This is analogous to indicating a point in time by saying "... now" instead of pointing "here". However, users did find that adding events while the tape was playing was difficult.

Inverting semantics of menu items

ConEd's marking menu permits a unique method for undoing. Commands can be undone in ConEd in the standard Macintosh manner (i.e., by pressing the "undo" key or selecting "undo" from the Edit menu). The limitation of this approach is that only the most recent edit can be undone. However, the laugh and ignore commands can also be undone by repeating the laugh or ignore command on the same event. The first laugh mark turns an event into a laugh event. A second laugh mark toggles the event back to a normal event. Therefore, even if these types of edits are not the most recent, they still can be undone.

Toggling the way the laugh and ignore commands work is an example of inverting a menu item semantics. In this case, once a function in a menu is invoked, it is replaced by the corresponding inverse function. Hence, the semantics are "inverted". For example, selecting "open" will invoke the open function and replace the "open" menu item with "close". There are several reasons why inverting semantics are important to marking menus. First, inverting semantics allows extra functions to be associated with a menu without increasing the number of items in a menu. This helps keep the number of items in a marking menu small, which in turn makes marking easier. Second, inverting semantics provides a mnemonic to help recall the association between mark and function. For example, if one remembers the mark associated with "open" then one can recall the mark associated with "close", because the two functions are the inverse of each other.

The role of command feedback

There are several ways that a user receives command feedback using marking menus in ConEd. When using the menu, the user knows which command is about to be executed because the name of the command appears highlighted in the menu. When marking, a user can either recall the mark/command correspondence or watch the results of drawing the mark. We have observed that, as users gain more experience with marking menus, they graduate from watching the menus and

90

marks, to watching the results of their actions to determine if they have selected the correct command.

Context also plays an important role in determining the command a mark triggers when semantic inversion is being used. For example, events that were marked as "laugh" events appeared in a gray color. This feedback provides essential information to the user that a "laugh" mark on this event was not actually a laugh command but a command to "unlaugh" the event.

In ConEd, a marking menu interaction combines object selection and command application. Typically, in mouse-based direct manipulation systems, these two actions are distinct. For example, a user selects an object by pointing to it; the object then appears "selected"; next, a command is applied to the object by selecting from a menu. When using the marking menu in ConEd objects never appear "selected". It is interesting to note that none of the users ever reported missing it. We can speculate the reason for this is that the combination of selection and marking is intuitive (i.e., emulates our experiences with pen and paper), and the result of a command appeared quickly enough that the starting point of the mark was still in visual image storage.

4.3. ANALYSIS OF USE

The behavior of two users using ConEd over an extended period of time was studied. Both users were employed to edit conversation data. The edited data was used in a research project which was independent of this research thesis. Therefore, a user's main motivation was not to use marking menus, but to complete the task of editing and coding the data. The amount of data to be edited was extremely large and therefore the users were mainly interested in performing the edits as quickly as possible.

The first user (user A) was an experienced Macintosh user and was also familiar with video technology. User A was also familiar with the intentions of the conversation analysis experiment. Given this profile, user A could be considered an expert, although unfamiliar at the start of the study with marking menus. The second user (user B) could be considered a novice. While user B did have some computer experience, it was mainly with the MS-DOS environment, not the Macintosh. Therefore, user B not only had to learn how marking menus worked,

91

she also had to learn the many details of the Macintosh interface, and the correct way to edit the conversation data.

It was explained to both users how the conversation data was to be edited. The goal of editing was to ensure that the data matched the conversation patterns on the video tape. Users edited the conversation patterns using ConEd and then checked their work by playing back the video tape and comparing the audio of the conversation with the data in ConEd. This process was very interactive. The user played the video and watched the conversation data "playback" on ConEd. When the user saw a piece of data that did not match the audio on the video tape, the user edited the data, then rewound and replayed the video tape and data to ensure the edit was correct.

Each user had the interface to ConEd explained to them and some example edits were performed for their benefit. In particular, the commands in the marking menu were carefully explained and demonstrated. The menu and mark mode was explained and demonstrated, as well as the ability to reselect menu items or confirm a mark. We then verified that the user understood the marking menu by having them perform a few edits using the menu and marks.

Data on user behavior was gathered by recording information about a marking menu selection every time a selection was performed. The information included the time the selection was made, the user's name, the item selected, the mode used to select the item (menu or mark), the length of time the selection took, and the path of the mark or the series of reselections from the menu. A user only needed to register his or her name at the start of an editing session. The rest of the trace data was accumulated transparently.

User A edited for a total of 8.55 hours over approximately six days. User B edited for 10.1 hours over a 29 day period. Most editing sessions lasted one to two hours.

After completing the task, the users were asked to fill out a questionnaire on their experiences using marking menus. The intention of the survey was to reveal users' perception of marking menus and gauge their level of satisfaction.

4.3.1. Issues of use and hypotheses

The main goal for tracing menu usage was to understand how users behave when using marking menus. Specifically, we wanted to find out whether or not in a real

92

work situation users would evolve from using the menus to using marks and the characteristics of this evolution. In Chapter 2, we described the design of marking menus and how it embodied several assumptions concerning user behavior. The assumptions are that, first, a user will begin by using the menu but with experience the user will evolve to using marks, and second, as part of this evolution, users will make use of intermediate modes of selection (i.e., mark-confirmation and reselection). We wanted to discover whether or not user behavior reflected this in order to prove our assumptions about the novice to expert transition, and to verify that these intermediate modes are actually needed in the marking menu design.

With these goals in mind, we formed the following hypotheses about user behavior with the marking menu in ConEd:

(1) Menu mode will dominate a user's behavior at first. However, with experience, mark mode will dominate.

(2) The more frequently a command is executed the more likely it is to be invoked by a mark.

(3) Users will make use of mark-confirmation and reselection but with experience this behavior will disappear.

The following hypotheses test our assumptions concerning the differences between novice and expert behavior. Specifically, expert behavior will demonstrate faster selection times and more efficient movement than novice behavior.

(4) Time to select from the menu, even with the wait delay subtracted, will be greater than time to make a mark.

(5) With experience, the average length of a mark and time required to make a mark will become smaller.

4.3.2. Results

We analyzed the data from the two users separately for several reasons. First, we were concerned with individual differences. Combining the data would have masked these differences. Second, this study was not a controlled experiment. The data being edited varied, as did the amount of time and number of sessions the users worked. Thus, there was no logical way to merge the users' trace data.

93

Finally, our two users were very different in attitude and expertise, and therefore combining the trace data would have been inappropriate.

Menu versus mark usage

Hypothesis (1) was shown to be true. Figure 4.9 shows the percentage of times a mark was used to make a selection (as opposed to using the menu to make a selection) versus the total number of selections performed. Over time, marking dominated as the preferred mode of selection. For user A, out of a total of 3,013 selections 6.6% used the menu. For user B, out of a total of 1,945 selections, 45% used the menu.

There are several interesting observations concerning the usage of marks over time. First, when users returned to using ConEd after a lay-off period, the percentage of marking dropped. Figure 4.10 shows that several long lay-offs from ConEd occurred during the study. Note the correspondence between periods of inactivity and dips in mark usage. This indicates that mark/command associations were forgotten when not practiced. However, the amount of fading reduced with the amount of experience (i.e., the dips in Figure 4.9 become less pronounced with experience). Second, note how user B's mark usage rises dramatically at approximately 650 selections. We believe the reason this happened was because user B was a very cautious and inexperienced user. For user B, every command was a new experience. For example, user B needed help opening, saving, and closing files. User B commented that it took her several hours to get comfortable with the video machine and the Macintosh interface before she could begin to think about using marks.

Hypothesis (2) claims that the more frequently a command is used, the more likely it will be invoked by marking. This is based on the assumption that frequent use demands fast interaction and this motivates a user to learn the association between mark and command. Some commands were used more frequently than others. The horizontal axes in the graphs in Figure 4.11 shows this. Hypothesis (2) is shown to be true by a strong correlation between the frequency at which a command was used, and the frequency at which that command was invoked by a mark. Figure

94

4.11 shows a linear relationship between frequency of command and frequency of marking (for user A, $r^2 = 0.81$, p<.05; for user B, $r^2 = 0.88$, p<.05)[15].

---

[15] Note that the add command was not used in this analysis because it appeared to be an outlier point. Its frequency of marking was much lower than the rest of the commands. Our users reported that the add command didn't work correctly all the time. Therefore we assume that users were not as confident about using a marking for the add command as they were for the other commands and hence the outlying mark frequency.

95
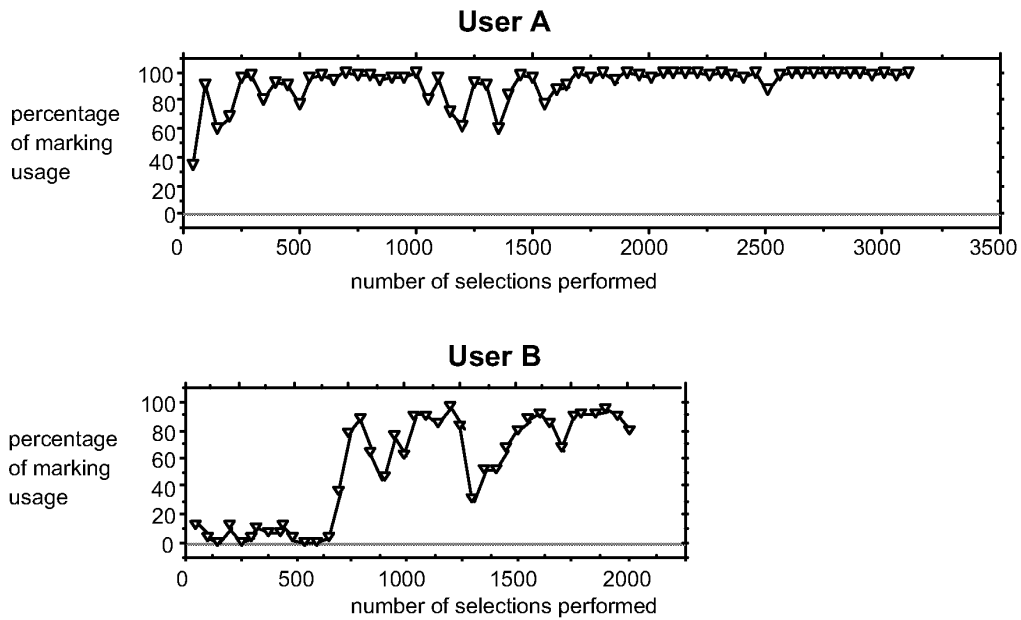
User A

User B

Figure 4.9: With experience, marking becomes the dominate method for selecting a command. Each data point is the percentage of times a mark was used in 50 selections.
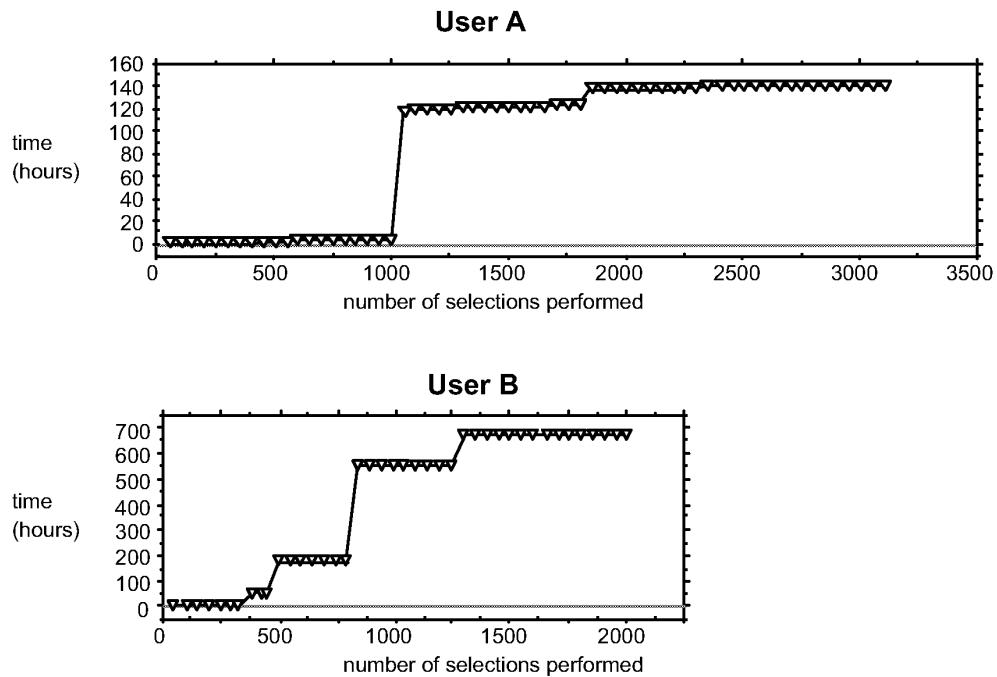
User A

User B

96

Figure 4.10: Usage of ConEd spanned many days with "lay-offs" between sessions. Steps in the graph represent layoff periods. Dips in the graphs in Figure 4.9 correspond to lay-offs. After a layoff, a user had to resort to the menu to reacquaint oneself with the marks (especially user B).
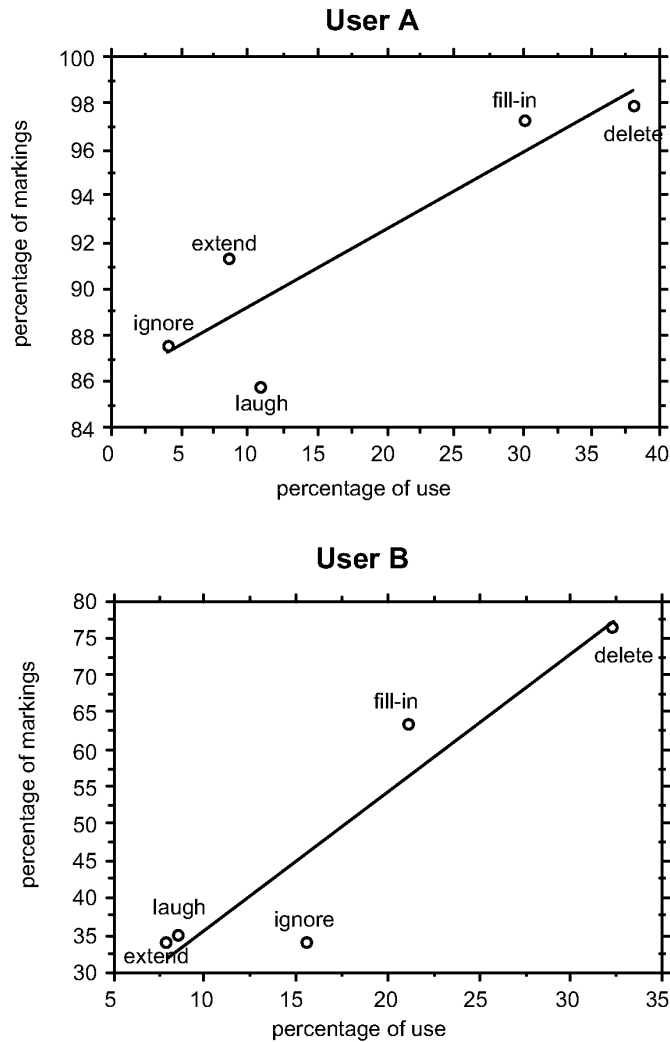

Figure 4.11: The more frequently a command is invoked the more likely it is to be invoked by a mark. The vertical axes show the percentage of times a mark was used to invoke a particular command. The horizontal axes show the percentage of times a particular command was invoked using either a mark or the menu.

97

Mark confirmation and reselection

As predicted by hypothesis (3), users did make use of the ability to confirm a mark and reselect from a menu but with experience this behavior disappeared. We draw evidence for this from Figure 4.12 as follows. Figure 4.12 (a) plots three types of behavior when using the marking menu:

- *mark*: a selection is made by making a mark;

- *mark-confirm*: a selection is made by making a mark but waiting at the end of the mark, thus popping up the menu to confirm the mark selects the correct item;

- *mark-corrected*: a selection is made in the same manner as "mark-confirm" but after popping up the menu another item is reselected.

We conjecture that these three behaviors are indicative of a user's skill in making accurate marks. Mark is the most skilled behavior. In this case, a user is so skilled at making a mark that no feedback is needed before confirming the selection. Mark-confirm is the next level of skilled behavior. In this case, a user has enough skill to make the correct mark but not the confidence to invoke it without checking it against the menu. Mark-corrected is a third level of skilled behavior. In this case, a user has made a mark, checked it against the menu and has corrected the mark using reselection.

Figure 4.12 shows several things. First, mark-confirm and mark-corrected behavior did occur and therefore this functionality is used and needed. Second, this behavior occurs during the transition from using the menu to drawing marks. Third, when used, this type of behavior occurred less than ten percent of the time.
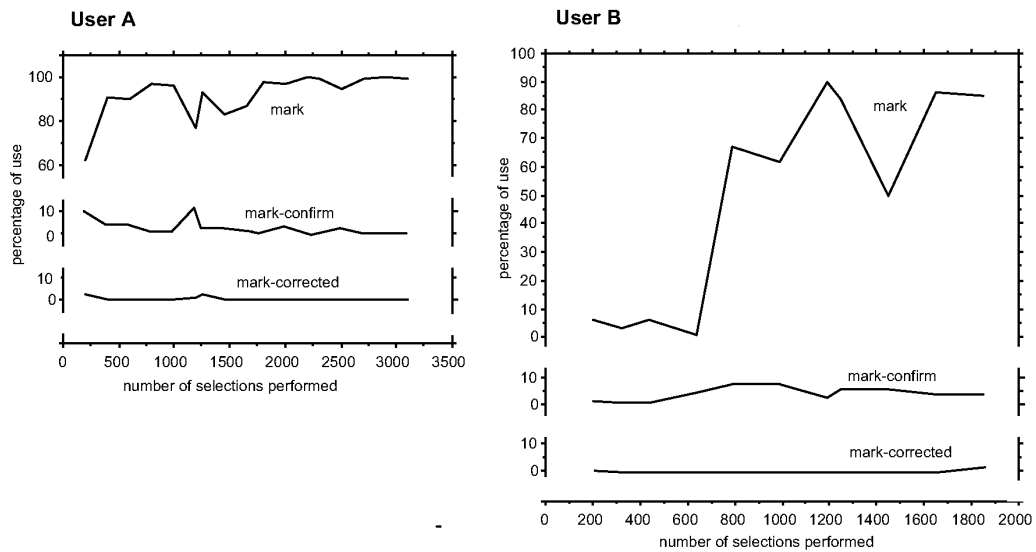
98

Figure 4.12: Users made use of the ability to confirm the selection a mark would make before committing to it. However, with experience this behavior disappears. Measures were averaged every 200 selections.

Reselection

Another topic of interest was whether or not users reselected when using menu mode. Figure 4.13 shows that reselection occurred less than ten percent of the time. User A demonstrated that with experience reselection disappears. However, user B did not exhibit this behavior. This is evidence that the reselection function in a radial menu is needed.
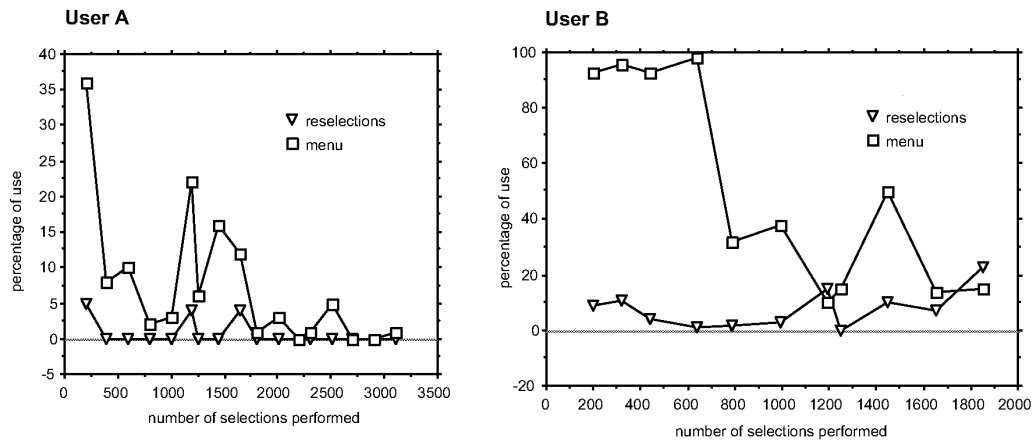
99

Figure 4.13: Both users utilized reselection in menu mode. While user A's use of reselection diminished with time, user B utilized reselection even after substantial experience. Measures were averaged every 200 selections.

### Selection time and length of mark

Selection time is defined as the time elapsed from the moment the mouse button is pressed down to invoke a marking menu, to the moment the button is released, completing the selection from the menu. This measurement applies to either a menu or mark mode. The selection time, for both users, was substantially faster in mark mode than in menu mode. Figure 4.14 shows these differences. For user A, a mark was seven times faster than using the menu. For user B, a mark was four times faster.

Even though menu and mark mode require the same type of movement, using the menu is slower than making the mark. There are several reasons why. First, a user must press-and-wait to pop up the menu. This delay was set to 0.33 seconds. However, as the fourth column in Figure 4.14 shows, even with this delay subtracted from the menu selection time, a mark is still much faster (i.e., user A is 4.2 times faster, user B is 3.0 times faster). The user most likely waits for the menu to appear on the screen. Displaying the menu takes the system about 0.15 seconds. The user must then react to the display of the menu (simple reactions of this type take no more than 0.4 seconds, according to Card, Moran, & Newell, 1983). However, when making a mark, the user does not have to wait for a menu to display and react to its display. Thus, a mark will always be faster than menu selection, even if press-and-wait was not required to trigger the menu. Figure 4.15 graphically shows this. The
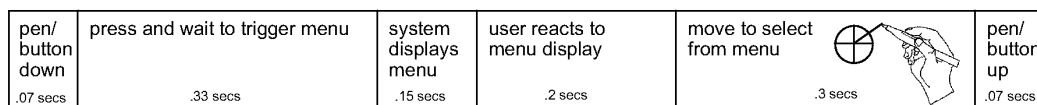
100

average time to perform a selection (seconds)		
	mark	menu	menu - delay
User A	0.18 ± 0.004	1.097 ± 0.042	0.763
User B	0.404 ± 0.01	1.543 ± 0.052	1.209

Figure 4.14: On average, marks were much faster than using the menu. For user A, a mark was seven times faster than using the menu. For user B, a mark was four times faster. Confidence intervals are at 95%.

Making a mark
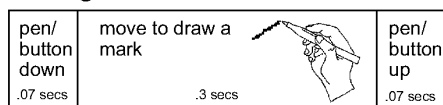
Using the menu

Time

Figure 4.15: Why a mark is faster than using a menu. The typical durations of various events that take place when making a selection are depicted. Even if press-and-wait was eliminated from menu selection it would still take longer than making a mark because of the additional events.

fourth column of Figure 4.14 provides evidence of this. This supports hypothesis (4).

Selection time, using a mark, decreased with practice, however the decrease was very small. In view of the very fast times for marking performance, this is good news, since this means that, even early in practice, novice performance was very similar to expert performance. The decrease in selection time was less than 0.1 seconds. For this analysis we used the Power Law of Practice (performance time declines linearly with practice if plotted in log-log coordinates (Snoddy, 1926)). Linear relationships were found for both users (an analysis of variance of linear

101

regression used; for user A, $F(1, 1654) = 166.5$, $p<0.0001$; for user B, $F(1, 541) = 23.03$, $p <0.0001$).[16]

The average length of a mark decreased slightly with practice for user B, but not for user A (an analysis of variance of linear regression used; $F(1, 2813) = 10.82$, $p<0.01$). The average length of a mark was approximately one inch. The delete mark was excluded from this analysis because its length was used to indicate a range of events.

Given these results for mark time and length we accept hypothesis (5)–mark time decreases with practice, but only in the case of user B is there support for the hypothesis that mark length also decreases with practice.

Users' perceptions

Both users were given a questionnaire after performing the editing task. The intention of the questionnaire was to discover if a user's perception of marking menus matched their behavior and also to allow us to obtain information not captured in the trace data.

An important parameter not captured in the trace data was selection errors. The reason for this is that prior to a selection we did not know what item a user intended to select. Therefore, when a selection was made, we could not tell whether or not the user had successfully invoked the desired selection. Since users should be the judges of what acceptable error rates are, we simply asked them how many errors they made with the marking menu: no errors, few but acceptable, or too many? Both users reported "few errors but acceptable".

Users perceived marking menus as a tool that helped them get the task completed quickly. Both users reported that their performance with the marking menu was "fast". User B, the less confident user, however, reported she didn't have enough regular experience with the marking menu to be completely comfortable drawing marks.

---

[16]   Linear relationships were determined by estimating a regression line using an analysis of variance approach (see Appendix A further explanation).

Both users confirmed the differences we found in performance between menu and mark mode. The trace data indicates that using a mark was substantially faster than using the menu. Both users reported a mark was "much faster" than using a menu.

We were also interested in how users recalled the relationship between command and mark. We suggested to both users three methods they could have used to recall mark/command associations. The first is by recollecting the spatial layout of the menu. The second is by rote–"this mark produces this command". The third method is the situation where one is so skilled at performing the mark/command that one is not aware of performing an explicit association—one just "does" the correct mark.[17] User A reported using the second technique, except in the case of "delete" for which he used the third method. User B reported using the first technique. If we assume that the three methods represent various stages of increasing practice, we can conjecture that user A was farther along in expertise and practice than user B. Our data shows this to be true (i.e., user A performed 1,068 more selections than user B).

Marking menus versus linear menus

The results from this study allow us to build on the comparison between marking menus and linear menus discussed in Chapter 2. When a user is familiar with the layout of a menu, selection from a radial menu will be faster than selection from a linear menu. Callahan et. al., (1989) provide empirical evidence of this for eight-item menus. It is possible that a linear menu may be more suitable when there an natural linear ordering to the menu items and a user must search the menu for an item before making a selection. Alternatively, a radial menu may be more suitable when there is a natural radial ordering of menu items. However, as shown by both Card (1982), and McDonald, Stone, & Liebelt (1983), the effects of organization disappear with practice. Callahan et. al., (1989) provide evidence that, for eight-item menus, even when menu items have a natural linear ordering, selection using a radial menu is still faster and less error-prone than selection using a linear menu.

Drawing from data in an experiment by Nilsen (1991), we can directly compare six-item marks and six-item pop-up linear menus. In Nilsen's experiment, a selection

---

[17] Recall may also be by rote in this case, but, since recall is so quick, users may perceive it differently.

from a six-item linear menu required on average 0.79 seconds. In our study, user A and user B required, on average, 0.18 and 0.40 seconds respectively to perform a selection using marks. Furthermore, in Nilsen's experiment the subjects' only task was to select from a linear menu. Therefore, one would expect selection speed to be artificially fast. In our study, in contrast, the users were performing selections in the context of other real world tasks.

The fact that radial menus are faster to select from than linear menus is not the complete story. Selection using a mark is faster than selection via a radial menu. This case study has shown marks to be substantially faster than selection from a radial menu, even if press-and-wait time is factored out. The reason for this is, when selecting using a menu, a user must react to the display of the menu before selecting. However, making a mark involves no reaction time. Hence selection with the mark is faster by design. Obviously faster selection with a mark comes at the price of higher error rates, especially when menus become dense. But the results from this chapter, Chapter 3, and Chapter 5 indicate that menus of breadths four, six and eight have acceptable error rates.

Thus, we can conclude that if menus contain an even numbers of items and less than ten of them, and users frequently use the menus, marking menus will have a distinct advantage over linear menus. Data from this chapter tells us that using the marks will be approximately 3.5 times faster than selecting from a radial menu. We conjecture this speed-up figure would be greater if compared to linear menus.

As a practical example of the impact of this speed-up, we can consider the performance of another real user using ConEd.[18] This user performed 16,026 selections during 36 hours of work. Her average time to select using a mark was 0.23 seconds. Her average time to select using the menu was 1.48 seconds. If the task had been done exclusively with a radial menu that did not use a press-and-wait delay of 0.33 seconds, the average time to select from a menu would have been 1.07 seconds, and 16,026 selections would have required 17,099 seconds in total. However, when using the marking menu, the menu was used for 185 selections and marks were used for 15,841 selections. Thus, menu selections required $185 \times 1.48 =$

---

[18] A third user used ConEd extensively over a long period of time but she was not included in this study because she assisted in the design of the marking menu used in ConEd and ConEd itself. Therefore, we felt she would not be an unbiased user of marking menus.

274 seconds. Selections made with a mark required 15,841 × 0.23 = 3627 seconds. This results in the 16,026 selections requiring 3901 seconds in total. Thus using a marking menu, as opposed to a radial menu that popped up immediately, saved the user 17,099 - 3,901 = 13,198 seconds or 3.66 hours.

4.4. SUMMARY

This chapter has described a case study which served two purposes. First, the case study involved designing an application that used a marking menu. From this exercise we gained insights on design. Second, data on two users' behaviors using this application to perform a real task was collected and analyzed. Information was collected on a user's performance using a marking menu every time a selection was performed. This information consisted of selection time, selection method, item selected, time of selection, and cursor movement. Analysis of this information allowed us to verify whether or not our assumptions about user behavior, which are embodied in the design of marking menus, are true.

This study demonstrated several things:

• A marking menu was a very effective interaction technique in this setting. Its effectiveness was contingent on applying the technique to an appropriate setting— specifically, using a marking menu to invoke a few commands that are used frequently, and require a screen location as a command parameter. Also, despite the simplicity of the mark, features of the mark, such as the start and end points, and the orientation of the mark, can be used to make interactions more efficient and easier to learn.

• A user's skill with marking menus definitely increases with use. A user begins by using the menu, but, with practice, graduates to making marks. Users reported that marking was relatively error free and empirical data showed marking was substantially faster than using the menu.

• The various modes of a marking menu (menu, mark, mark-confirmation, and reselection) are utilized by users and reflect levels of skills. In addition, when a user's skill depreciates during a long lay-off period, the user utilizes these modes to reacquire skills. We conclude that these features are a necessary part of the design,

105

and furthermore, interfaces which supply mutually exclusive novice and expert modes are inappropriate when a user's level of skill depreciates.

• In this setting a mark is a very fast way to invoke a command, and users, very early in practice, become skilled at making marks. Evidence of this is that selection time was much faster in mark mode than in menu mode, and did not decrease substantially with practice. This same data indicates that even if the delay time is removed from a menu selection time, menu selection is still slower than marking. This may be due to a user simply moving slower when using the menu. In theory, however, even if there was not press-and-wait delay, and the user moved as quickly in menu mode as they do in mark mode, the user would still be delayed by, first, having to wait for the system to display the menu, and, second, by their own reaction time to its display. Hence, within the limitations on the number of items in a menu described in Chapter 3, we conclude that a mark will always be faster than a menu that immediately pops up. This, of course, is dependent on the user recalling the menu layout.

We can expect hierarchic marking menus to exhibit the same performance properties as non-hierarchic marking menus, since selection from a hierarchic marking menu consists of a series of selections from non-hierarchic menus. Chapter 5 establishes the breadths and depths of hierarchy at which we can expect these properties to hold true.

106

Chapter 5: An empirical evaluation of hierarchic marking menus

This chapter reports on an experiment to investigate the characteristics of human performance with hierarchic marking menus. Performance using a hierarchic marking menu is affected by the number of items in each level of the hierarchy and the depth of hierarchy. This chapter reports on an experiment which systematically varied these parameters to determine the conditions under which using a mark to select an item becomes too slow or prone to errors. Increasing depth and breadth tends to degrade performance. Thus the intention of this experiment was to find an practical upper bound for these parameters. Understanding of the role of depth and breadth helps us address the types of questions one asks when designing hierarchic marking menus for an interface:

Q1: Can users use hierarchic marks? Chapters 3 and 4 have shown non-hierarchic marking menus to be useful. (Hopkins, 1991) describes how hierarchic pie-menus can be useful. Thus we can expect hierarchic marking menus, even without using marks, to also be useful. However, the question remains: Is it possible to use a mark to quickly and reliably select from a hierarchic radial menu?

Q2: How deep can one go using a mark? Just how "expert" can users become? Can an experienced user use a mark to select from a menu which has three levels of hierarchy and twelve items at each level? By discovering the limitations of the technique we are able to predict which menu configurations, with enough practice, will lead to reliable selections using marks, and which menu configurations, regardless of the amount of practice, will never permit reliable selections using marks. Also, will some items be easier to select regardless of depth? For example, it

107

seems easier to select items that are on the up, down, left and right axes even if the menus are cluttered and deep.

Q3: Is breadth better than depth? Will wide and shallow menu structures be easier to access with marks than thin and deep ones? Traditional menu designs have breadth/depth tradeoffs (Kiger, 1984). What sort of tradeoff exists for marking menus?

Q4: Will mixing menu breadths result in poorer performance? The experiment on non-hierarchic marking menus described in Chapter 3 has shown that the number of items in a menu and the layout of those items in the menu affects subjects' performance when using marks. Specifically, menus with 2, 4, 6, 8 and 12 items work well for marks. What will be the effect of selecting from menu configurations where number of items in a menu varies from sub-menu to sub-menu?

Q5: Will the pen be better than the mouse for hierarchic marking menu marks? The experiment in Chapter 3 compared making selections from non-hierarchic marking menus using a stylus/tablet, a trackball and a mouse. Subjects' performance was poorest with the trackball while performance with the stylus/tablet and mouse was approximately equal. However, hierarchic marking menus require more complex marks. Will the mouse prove inadequate?

We are also concerned with some pedagogical issues which help us design human-computer interactions. Buxton has described the notion of *chunking* in human-computer dialogs (Buxton, 1986). For example, when using a mark to specify a "move" command, one can issue the command verb, source and destination all in one mark or "chunk". This notion is related to the concept of a "motor program" in motor control studies. A motor program is "a set of muscle movements structured before a movement begins, which allows the entire sequence to be carried out uninfluenced by peripheral feedback" (Keele, 1968).

Some systems or interaction techniques allow chunking to take place while others don't. In some systems a user can articulate a series of operations without having to wait for the system to finish each operation. This allows these commands to be chunked. For example, a user quickly clicks on three graphical buttons without having to wait for each button to complete its operation. In this case, the user may perceive the three clicks as a chunk. If the user was restricted to wait for each button to complete its operation before clicking on the next button, the user may not

perceive the three operations as chunk. Hence, this indicates that something as low-level as input event handling policies can affect user perception and behavior.

Relative to marking menus, the phenomenon of chunking occurs when a user, rather than articulating a selection from a hierarchic menu as a series of directional strokes separated by pauses in movement, performs the entire series of selections in one fluid movement or "chunk". We speculate that chunking is related to expertise. The more expert a user becomes with an interface the more the user chunks. This experiment provides the opportunity to investigate this phenomenon.

5.1. THE EXPERIMENT

5.1.1. Design

In order to determine the limits of performance, we needed to simulate expert behavior. We defined expert behavior as the situation where the user is completely familiar with the contents and layout of the menu and can easily recall the mark needed to select a menu item. To make subjects "completely familiar" with the menu layouts we chose menu items whose layout could be easy memorized. We tested menus with four, eight and twelve items. For menus of four items, the labels were laid out like the four points of a compass: "N", "E", "S" and "W". This type of menu we referred to as a "compass4". Similarly, a "compass8" menu had these four directions plus "NE", "SE", "SW" and "NW". Menus with twelve items, referred to as a "clock" menus, were labeled like the hours on a clock.

Will users of real applications ever be as familiar with menus as they are with a clock or compass? We believe the answer is yes, and base this on three pieces of evidence. First, our own behavioral study of users using a marking menu in a real application (Chapter 4) shows, with practice, they used marks without the aid of menus over ninety percent of the time. Other researchers have reported this type of familiarity with pie menus (Hopkins, 1991). Second, Card (1982), and McDonald, Stone, & Liebelt (1983) report that effects of menu organization disappear with practice. In other words, with practice, users memorize menu layouts and navigate directly to the desired menu item. Finally, it must be remembered that a user does not have to memorize the layout of an entire menu. For example, a hierarchic

marking menu could contain 64 items but the user might only memorize the marks needed to select the two most frequently used menu items.

The design of a trial in our experiment was as follows. A subject was completely familiar with the menu layout and the marks needed to select an item. The system would ask the subject to select a certain item using a mark (the menu could not be popped up by the subject). The subject would input the mark and the system would then record the time taken to draw the mark and whether or not the mark successfully selected the requested item. After a series of trials, we would then vary the menu configuration and input device in order to see what effect these variables had on selection performance.

The rationale for choosing menus of four, eight and twelve items was based on the results from the experiment in Chapter 3. This experiment showed that menus with even numbers of items and less that twelve items were suitable for marking. Using four, eight and twelve item menus is a deliberate attempt to explore a reasonable range of menu breadth. We would expect that performance on a menu of four items to be quite acceptable even at extreme depths. Whereas selection from a menu structure consisting of twelve-item menus which are two levels deep, seems quite treacherous.

Using a similar rationale, we chose to evaluate menu depths from one to four. A depth of one is a non-hierarchic menu which we know from the experiment in Chapter 3 produces acceptable performance. A maximum depth of four was chosen since it is in the range where we believe performance will become unacceptable.

For the sake of brevity we adopt a simple notation in the experiment. A menu structure can be described by a tuple B,D. B is the breadth of each menu in the structure and D is depth of menu structure. For example, 8,2 menu is a menu hierarchy where every menu contain eight items and the hierarchy is two level deep. An 8,2 menu contain 64 leaf menu items. When menu structures consist of different breadth at different levels we use the notation B:B:B, where B is the breadth of a menu at a certain depth. For example, a 12:8:12 menu is menu structure consisting of a top level menu of twelve items, second level menus of eight items and a third level menus of twelve items. A 8,2 menu is represented in this notation as 8:8.

In menu structures of even moderate depth and breadth the number of selectable items becomes very large. For example, in an 8,2 menu there are 64 selectable items.

110

As stated earlier, we wanted to simulate the case where the user was familiar with the marks being drawn. Given the practical time constrains of the experiment we could not expect subjects to become familiar all marks. Instead we decided to use only three target selections for each menu structure. A subject could then quickly become familiar with the mark needed to make the target selection with a reasonable amount of practice. In this way, the experiment addressed the question: given that the user knows the mark and is practiced at making it, will selection be quick and reliable?

The next issue concerning targets was "which three targets"? For menus of small breadth and depth this was not a major issue as one type of selection is approximately as easy to draw as another. However, in the case of menus which consist of combinations of larger breadths and depths, some selections are definitely harder that others. For example, we observed that making the selection "12-6-3-9" from a 12,4 menu was much easier that "10-11-10-11".

Our approach was to pick three targets for each menu configuration such that one was easy, one was moderately difficult, and one was difficult. Easy targets were those that had items along the vertical and horizontal axes (on-axis items). Difficult targets were those with items not on the vertical and horizontal axes (off-axis items), and little angle change between items. Finally, targets of moderate difficulty were those with a 50% mix of on-axis and off-axis items, and a 50% mix of little and large angle changes between items. It was hoped this mixture of targets would result in behavior that would be representative of an "average selection".

In the case of menus that contain only on-axis items and large angle changes, we observed, prior to the experiment, that up and to the left selections seemed to be hardest, and down to the right selections seemed to be easiest. Thus we chose hard targets and easy targets accordingly. For moderately hard targets we chose either down-and-to-the-left, or up-and-to-the-right targets.

We also had subjects perform selections from a 12:8:12 menu. This was done so we could observe the effects of combining menus of different breadths in a menu configuration.

5.1.2. Hypotheses

Nine hypotheses are proposed:

111

(1) Pen outperforms mouse: The subjects will perform better with the pen than with the mouse in terms of response time and errors. Once again, the experiment in Chapter 3 showed that subjects performed marginally better with the stylus/tablet than with the mouse on non-hierarchic marking menus. However as depth increases, marks become more complex to draw and therefore the pen should be a more suitable device.

(2) Increasing breadth increases response time and errors: As breadth increases, response time and error rate will increase. The experiment in Chapter 3 demonstrates this effect for non-hierarchic marking menus. Therefore, we believe this effect will apply to hierarchic marking menus as well.

(3) Increasing depth linearly increases response time: As depth increases response time will increase. We base this on the belief that marks to access deep menu configurations will require more time to draw because they will be longer.

A study by Fischman is the most relevant work to this hypothesis (Fischman, 1984). In the study, subjects used a stylus to tap on a series of metal disks (ranging from one to five disks) that were either arranged in a straight line or in a staircase pattern that required changes in direction of 90û between disks. Changes in direction and different numbers of disks in the series roughly correspond to directional movements and different depths in hierarchical marking menus. Fischman found that response time linearly increased with the number of disks, but changes in direction did not affect response time.

(4) Increasing depth increases errors: As depth increases error rate will increase. As depth increases so does the number of times a subject has to estimate at the angle of mark needed to select an item. Therefore the error rate will increase as the probability of error increases.

(5) Inaccuracies propagate: We hypothesize that the depth at which errors take place will be on average greater than half the depth of a menu structure. Informally, we claim that inaccuracies in one portion of a mark will affect the accuracy of the remaining portion of a mark. Our reasoning is as follows: a subject uses the angle of a partially drawn mark to estimate the angle for the next portion of the mark. Inaccuracy in the first portion of the mark may then propagate into the rest of the mark, eventually resulting in an error. The effect of this is that the probability of an error increases with depth. If the probability of an error was the same at every level,

an error would occur on average at half the depth of the menu structure. However, if the probability of an error increases with depth we should see errors take place on average at a depth greater than half the depth of the menu structure.

(6) Mixing menus degrades performance: Combining menus of different breadths in a menu configuration will degrade response time and increase the error rate relative to menu configurations where all menu breadths are the same. Specifically, we hypothesize that subjects will perform better on a 12:12:12 menu than on a 12:8:12 menu, even though an eight-item menu is easier to select from than a twelve-item menu. We believe it is easier for users to select items when the difference of stroke angle needed to select different items is consistent. For example, in a menu structure consisting exclusively of eight-item menus, all items are at 45 degree angles. If a twelve-item was introduced into the menu structure, some items would be at 45 degrees while others, the ones from the twelve-item menu, would be at 30 degree angles. We believe inconsistency in "item angle" will degrade performance.

(7) On-axis items enhance performance: Marks that consist of on-axis items will be faster to draw and produce fewer errors than marks that consist of off-axis marks. This hypothesis is based on prior practical experiences using hierarchic marking menus.

(8) Drawing direction affects performance: The direction of drawing will affect performance. Specifically, marks that require drawing left to right will be performed faster than marks that require drawing right to left. Other researchers have found a similar bias in directional movements (Boritz, Booth, & Cowan, 1991; Malfara & Jones, 1981; Guiard, Diaz, & Beaubaton, 1983).

(9) Subjects will chunk: The number of pauses when drawing a selection will approach zero with practice. Once a subject starts to think of selection not as a series of strokes at certain angles, but as a mark of a certain shape, the subject will draw the mark without pauses between strokes. This hypothesis was based on our own experiences using marking menus in the laboratory.

5.1.3.   Method

Subjects: Twelve right handed subjects were recruited from University of Toronto. All subjects were skilled in using a mouse but had little or no experience using the pen on a pen-based computer.

Equipment: A Momenta pen-based computer development system was used. The input devices consisted of a Microsoft mouse for IBM personal computers, and a Momenta pen and digitizer. The digitizer was transparent and placed over the screen. This allowed subject to "write on the screen" with the pen. The screen was placed in front of the subject at approximately a 45 degree angle. When using the pen the hand could be rested on the screen. The mouse was placed to the right of the screen on a mouse pad. No mouse acceleration was used and the sensitivity of the mouse was set to a value of 50 in the control panel. A setting of 50 corresponds to a one to one C:D ratio.[19]

Task: A trial occurred as follows. The type of menu structure being tested appears in the top left corner of the screen. A small circle appears in the center of the screen. A subject then presses and holds the pen or mouse button over the circle. The system then displays instructions describing the target at the top center of the screen. A subject then responds by drawing a mark that is hoped to be the correct response. The system responds by displaying the selection produced by the mark. If the selection did not match the target, the system beeps to indicate an error. The system then displays each menu in the current menu structure at its appropriate location along the mark and indicates the selection from each menu. The subject's score would be shown in the lower left of the screen. Figure 5.1 shows the experimental screen at this point. If the selection is incorrect, a subject loses 100 points and the trial is recorded as an error. If the selection is correct, the subject earns points based on how quickly the response was executed. Response time is defined as the time that elapsed between the display of the target and the completion of the mark.

A subject's score (accumulated points) is displayed in the lower left corner of the screen plotted against current trial number. The graph also shows the best score for that particular pairing of menu structure and input device. This gives subjects a performance level to compete against. This helped to ensure that subjects performed the task both quickly and accurately.

A subject's progress through the trials was self paced. Subjects could pause between trials for as long as they liked. Subjects used this pause to check their score and rest.

_____

[19] See Section 2.5.3 for the definitions of C:D ratio and mouse acceleration.

Most subjects paused just a few seconds. All subjects required approximately one hour and fifteen minutes to complete the experiment.
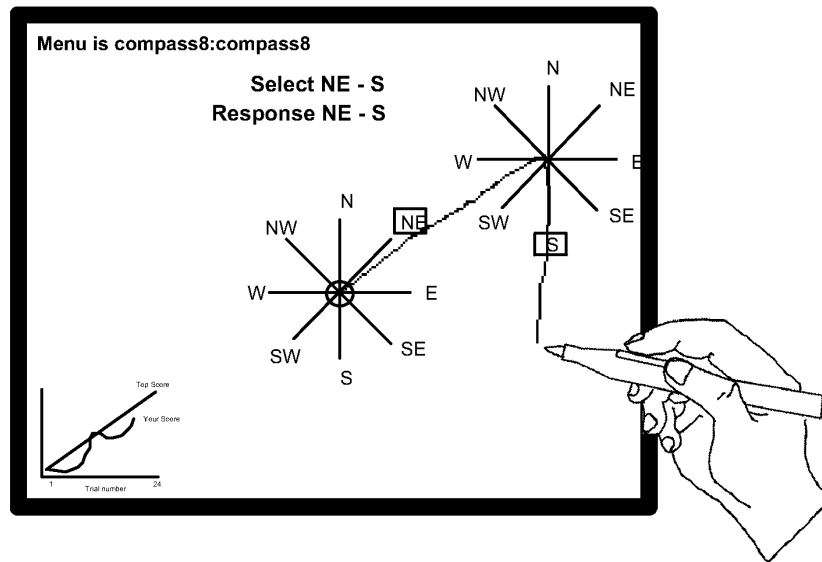

Figure 5.1: The experiment screen at the end of a trial where the target was "NE-S". After the mark is completed, the system displays the menus along the mark to indicate to the subject the accuracy of their marking.

Design: All three factors, device, breadth and depth were within-subject. Trials were blocked by input device with every subject using both the pen and the mouse. One half of the subjects began with the pen first while the other half began with the mouse. For each device, a subject was tested on the thirteen menu structures (breadths 4, 8 and 12 crossed with depths 1 to 4, plus the mixed menu structure of 12:8:12). Menu structures were presented in random order. For each menu structure a subject performed 24 trials. For the 24 trials, subjects were repeatedly asked to select one of three different targets. Each target appeared eight times in the 24 trials but the order of appearance was random.

Given this design, for each data point (a particular combination of input device and menu structure) 288 selections were collected (24 selections times 12 subjects). For the experiment, 7,488 selections were performed in total.

115

Before starting a block of trials for a particular menu configuration, subjects were allowed eight seconds to study the menu configuration. Before starting trials with a particular input device, a subject was given ten practice trials using the device on a 4,3 menu. This was intended to acquaint a subject with the "feel" of the input device.

It can be argued that the practice session on the 4,3 menu gave subjects an unfair advantage on this particular menu. We believe the effect was small for several reasons. First, a different set of targets was used for practice than those used in the timed trials so subjects did not become practiced at drawing the targets for the timed trials. Second, because of our choice of obvious menu labels and structure for all menus, a subject was already familiar with all of the menu structures even before practice.

5.2.   RESULTS AND DISCUSSION

The main dependent variables of interest were response time and percentage of errors. Response time was defined as the time that elapsed between the display of the target and the completion of the mark. Percentage of errors was the percentage of incorrect selections out of 24 trials on a particular combination of device, breadth and depth. Figures 5.2 and 5.3 show the means tables.

Response time averaged across all subjects, breadths and depths for the pen was 1.69 seconds, while the mouse was significantly slower at 2.07 seconds $(F(1,11)=19.7, p < .001)$. The subjects produced significantly more errors with the mouse than with the pen $(F(1, 11)=6.41, p < .05)$. Subjects' performance with the pen was better than with the mouse both in terms of response time and percentage of errors, and therefore we accept hypothesis 1.

Breadth significantly affected both response time $(F(2,22)=91.7, p < .001)$ and errors $(F(2,22)=130.5, p < .001)$. Figure 5.3 shows, in general, that increasing breadth increases response time and percentage of errors. Based on these results we accept hypothesis 2.

Depth significantly affected both response time $(F(3,33)=195.4, p < .001)$ and errors $(F(3,33)=51.5, p < .001)$. Figure 5.3 (a) shows a linear increase in response time as depth increases. Linear regression on each device, menu breadth pair verifies this

116

claim (for the pen: breadth four, $r^2 = 0.79$, breadth eight, $r^2 = 0.88$, breadth twelve, $r^2 = 0.82$; for the mouse: breadth four, $r^2 = 0.73$, breadth eight, $r^2 = 0.77$, breadth twelve, $r^2 = 0.67$; $p < .001$ for all values). Figure 5.3 (b) shows that as depth increased so did percentage of errors. Given these results we accept hypotheses 3 and 4.

Breadth	Device	Depth					mouse & pen
		1	2	3	4	Total	
4	mouse	.752 (.146)	1.189 (.188)	1.797 (.407)	2.102 (.445)	1.460 (.616)	1.367 (.554)
	pen	.710 (.108)	1.098 (.142)	1.451 (.275)	1.835 (.298)	1.279 (.473)	
8	mouse	.932 (.211)	1.665 (.543)	2.938 (.829)	3.309 (.845)	2.211 (1.159)	2.021 (1.047)
	pen	.810 (.142)	1.411 (.298)	2.258 (.420)	2.843 (.698)	1.831 (.895)	
12	mouse	1.170 (.289)	1.842 (.407)	3.011 (.763)	4.181 (1.363)	2.551 (1.406)	2.250 (1.208)
	pen	.915 (.236)	1.531 (.266)	2.331 (.519)	3.022 (.443)	1.950 (.888)	
Total	mouse	.951 (.278)	1.565 (.484)	2.582 (.877)	3.197 (1.272)	2.074 (1.194)	
	pen	.812 (.186)	1.347 (1.272)	2.013 (.572)	2.567 (.724)	1.685 (.826)	
	mouse & pen	.881 (.245)	1.456 (.415)	2.298 (.789)	2.882 (1.075)		

Figure 5.2: Means table for response time. Each entry is average response time in seconds. Standard deviation is shown in parentheses.

117

| Breadth | Device | Depth | | | | | mouse & pen |
		1	2	3	4	Total	
4	mouse	1.43 (2.81)	4.18 (4.35)	4.24 (3.59)	5.10 (4.20)	3.74 (3.92)	3.94 (4.65)
	pen	2.19 (5.09)	4.59 (4.63)	4.91 (5.97)	4.89 (5.69)	4.15 (5.32)	
8	mouse	5.90 (4.85)	8.82 (4.62)	20.44 (8.60)	22.98 (9.64)	14.51 (10.18)	12.42 (9.93)
	pen	5.21 (7.13)	6.64 (6.56)	16.71 (9.84)	12.77 (9.26)	10.33 (9.34)	
12	mouse	13.19 (6.37)	21.26 (8.79)	38.56 (14.98)	38.58 (12.06)	27.90 (15.45)	24.50 (16.26)
	pen	8.33 (8.33)	14.61 (14.91)	31.87 (14.13)	31.87 (14.13)	21.09 (16.48)	
Total	mouse	6.84 (6.84)	11.42 (9.51)	21.08 (17.32)	22.19 (16.52)	15.38 (14.69)	
	pen	5.24 (7.24)	8.62 (10.46)	17.83 (15.5)	15.74 (14.90)	11.86 (13.29)	
	mouse & pen	6.04 (7.04)	10.12 (10.02)	19.46 (16.24)	18.96 (15.95)		

Figure 5.3: Means table for percentage of errors. Standard deviation is shown in parentheses.
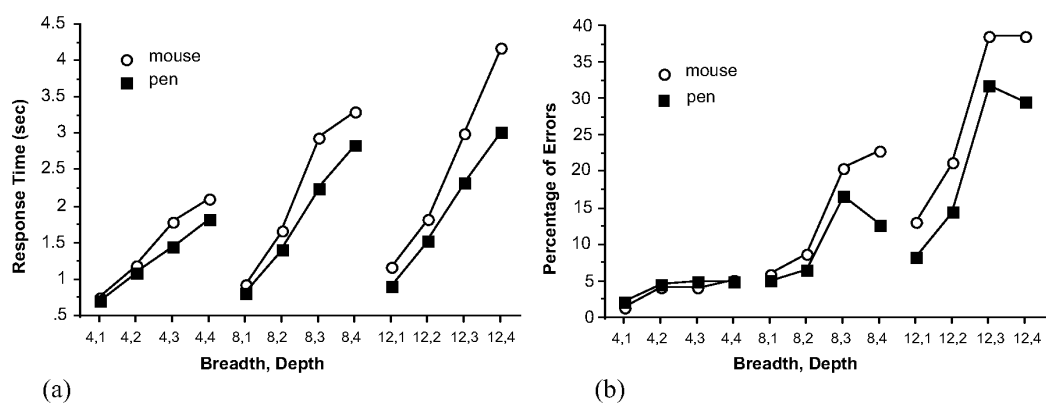

Figure 5.3: Response time and percentage of errors as a function of menu breadth, depth and input device. Each data point is the average of 288 trials.

All three factors, input device, breadth and depth affected response time. Analysis of variance revealed a three way interaction between input device, breadth, and depth ($F(6,66)=3.32$, $p < .05$) affecting response time. Figure 5.3 (a) shows these relationships. As one would expect, increasing breadth and depth increases

response time, however subjects' performance degraded more quickly with the mouse than with the pen.

Both depth and breadth interacted to affect error rate ($F(6,66)=12.28$, $p < .0001$). Variance in the error data is large, so the curves in Figure 5.3 (b) must be interpreted carefully. Individual comparisons of error means revealed no significant differences for breadth four at any depth. For breadth eight and twelve, the only significant change in error rate occurred between depth two and three ($F(1, 11) = 23.85$, $p < .001$; $F(1, 11) = 60.52$, $p < .0001$). This indicates that the "rolling off" of the errors curves for breadths eight and twelve between depths two and three is not statistically significant but the increase between depths two and three is significant.

It is important to compare these errors against what we believe would be reliable menu configurations. It seems reasonable that selection from 4,2 menus would be reliable since these marks can be recognized even if drawn very inaccurately. A comparison between 4,2 and 8,2 menus reveals no significant difference. Hence, we have no evidence to claim that eight-item menus, up to two levels deep are more unreliable than 4,2 menus.

A similar comparison between the 4,2 and 12,1 menu revealed a significant difference ($F(1, 11) = 8.25$, $p < .01$). However, the 12,1 menu was not significantly different from the 8,2 menu. Continuing the comparison, we found that the 12,2 menu was significantly different from the 8,2 menu ($F(1, 11) = 21.11$, $p < .0001$). Hence, we claim that the 12,1 menu borders on being unreliable. Section 5.2 has further interpretations on these results.

Hypothesis 5 (inaccuracies propagate) was shown to be true. As depth increased, the average depth at which errors occurred became significantly greater than half the depth of the hierarchy ($F(3,33)=7.62$, $p < .001$). However, the input device had an effect on this behavior. Figure 5.4 shows the pen consistently demonstrated this effect but the mouse exhibited a more erratic behavior.
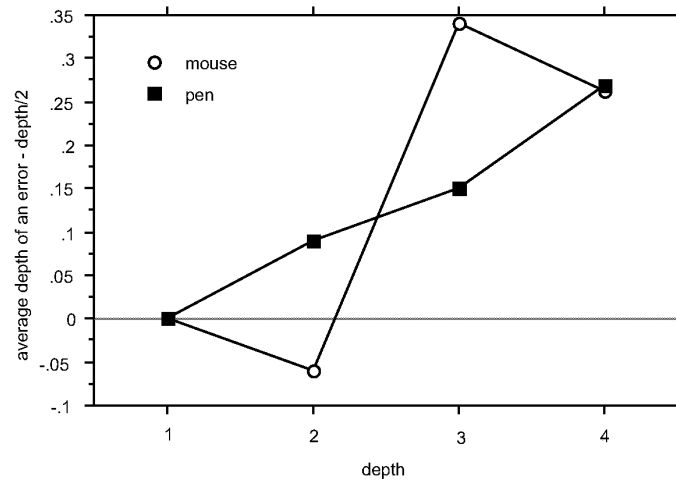
119

Figure 5.4: Average depth of error - depth/2 versus depth. Depth is the depth of the menu structure being selected from.

We tested the effects of mixing menu breadths in menu configurations by comparing the performance of a 12:12:12 menu with a 12:8:12 menu. We found no significant performance difference between the two menu structures. Therefore, we have no evidence that hypothesis 6 (mixing menus degrades performance) is true.

In order to test the hypothesis 7 (on-axis items enhance performance), targets for the 12,2, 12,3 and 12,4 menus were picked such that the experiment data could be divided into 3 groups. With each group we associated an "off-axis-level": a1, a2 and a3. Experimental data was placed in group a1 if the target consisted only of menu items that were on-axis, such as "12-3-9-3." Group a3 consisted of data on targets that consisted of entirely off-axis targets such as "1-2-1-2". Group a2 consisted of data on targets that were a mixture of on-axis and off-axis menu items, such as 12-7-3-9. Figure 5.5 shows that axis level had a significant effect on response time $(F(2,22)=104.84, p < .001)$, and on percentage of errors $(F(2,22)=36.2, p < .001)$. Figure 5.5 (a) shows how the type of device interacted with off-axis level $(F(2,22)=6.93, p < .05)$. This indicates that subjects response time using the pen did not degrade as much as their response time with the mouse on the worse off-axis targets.
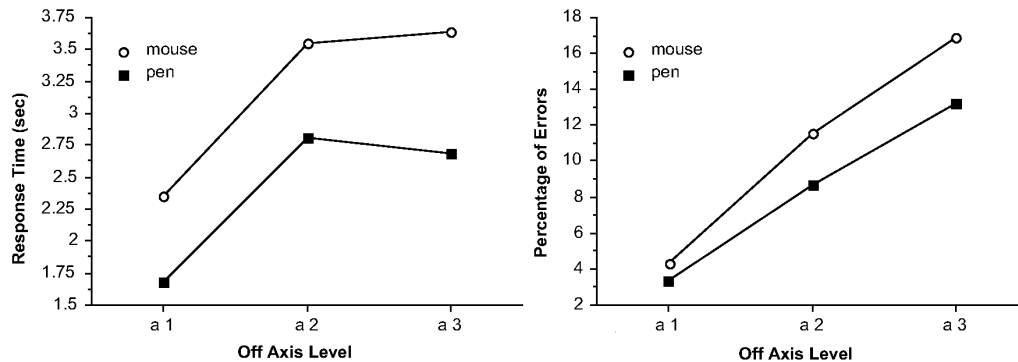
120

Figure 5.5: Average response time and percentage of errors for targets with an increasing number of "off-axis" items.

In order to evaluate hypothesis 8 (drawing direction affects performance), targets were picked for 4,3, 4,4, 8,3, and 8,4 menus such that mirror image pairs of targets could be compared. For example, the target N-W-N-W was compared with the target N-E-N-E. No significant different in response time was found between "left" and "right" direction targets. Therefore this experiment provides no evidence that hypothesis 8 is true.

The data was analyzed for learning effects by examining performance after every sixth trial. Figure 5.6 shows the results. Response time dropped over 24 trials ($F(3,33)=59.227$, $p<.0001$). Percentage of errors dropped as well ($F(3,33)=8.294$, $p<.0003$). This shows that not only were subjects getting faster but also producing fewer errors. No significant performance differences were found between trial 18 and trial 24. It may be possible that, because subjects were only selecting from three targets, their performance was beginning to asymptote by trial 24.
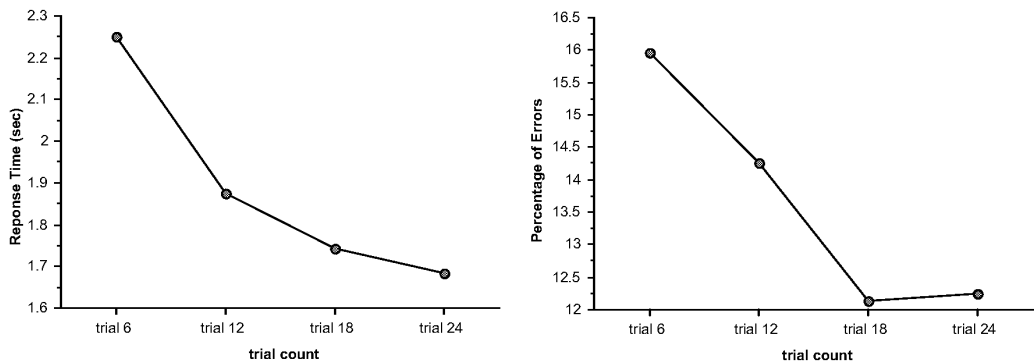
121

Figure 5.6: Average response time and percentage of errors after every sixth trial.

We analyzed the data for the number of pauses that occurred as a selection was being drawn. A pause was defined as the pen or mouse not moving more than five pixels for more than 1/2 of a second. Figure 5.7 shows that, as users gained experience the number of pauses dropped $(F_{(9,99)}=38.409$, $p < .0001)$. This is evidence that subjects, with experience, began to draw a mark not as a series of discrete selections but as a single mark of a certain pattern (assuming that when pauses did occur they occurred between different selections). The number of pauses did not fall all the way to zero because some of the most difficult targets required careful drawing which resulted in pauses. Given these results we accept hypothesis 9 (subjects will chunk).
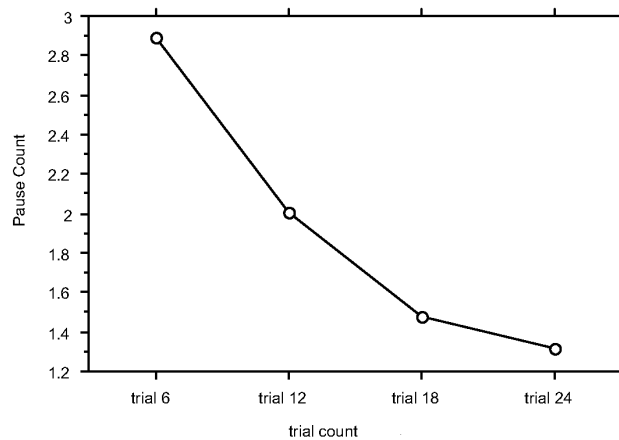

Figure 5.7: Average number of pauses counted after every six trials.

122

We also gave subjects a questionnaire after the experiment. This was to elucidate subjects' perception of their own performance and compare some of the experimental data with subjects' perceptions.

Eleven out the twelve subjects thought making selections with the pen was faster and more accurate than with the mouse. This agrees with the data from the experiment. Also, when asked to comment on the experiment, four subjects reported that, although their performance with the mouse was fast, they found the mouse required more effort.

We wanted subjects' opinion on the accuracy of our mark recognizer. In some cases, for example, menus which are only one level deep, recognition is simple. In this case, only the start and end points of the mark need to be examined to determine the item picked. However, at depths greater than one, submenu selections must be determined so changes in direction along a mark must be recognized. The algorithm for determining these "kinks" along a mark is complex because it has to handle dense menus and marks that are drawn sloppily. Typical of most recognition systems, occasionally what appears to the subject as a correct mark is misinterpreted by the system. However, on average, subjects claim that this happened only three percent of the time. This is an acceptable recognition rate by mark recognizer standards (Sibert, Buffa, Crane, Doster, Rhyne, & Ward, 1987). Nonetheless, after observing the type of recognition errors that occurred during the experiment, we believe the recognition rate can be further improved by a few refinements to the recognizer algorithm.

Another phenomenon that occurred in the experiment was subjects selecting the wrong direction by accident. For example, the screen would display "select N" and the subject would select south. Errors of this type are referred to as "mental slips" (Norman, 1981). These types of errors were removed from the data set before analysis because they are not caused by drawing inaccuracies. Other errors such as clear cut errors on part of the recognizer were also removed from the data. Subjects reported several causes for mental slips: "I just goofed" or "I started to draw the mark from the previous trial". Subjects, on average, claimed that mental errors occurred two percent of the time. This approximately agrees with the data: we found a one percent error rate for clear cut "mental slips". We do not feel these

123

errors are particular to drawing marks—mental slips are common in any human activity (Norman, 1981).

We hypothesized before the experiment that drawing marks that were predominately left to right movements would be easier, and hence faster, than right to left marks. However, our analysis of the data showed drawing direction had no significant effect on selection times. This agrees with the results of the questionnaire: six out of the twelve subjects thought left to right marks were easier to draw that right to left marks. This even split among subjects perhaps explains the non-significant effect of direction. A closer examination of the data might reveal individual effects.

5.3.   CONCLUSIONS

We can now revisit the questions posed at the start of this chapter and interpret the results of this experiment.

Q1: Can users use hierarchic marks? Even if using a mark to access an item is too hard to draw or cannot be remembered, a user can perform a selection by displaying the menus. Nevertheless, since the subjects could perform some of the marks in the experiment with acceptable response times and error rates, marking is a usable method of selection.

Q2: How deep can one go using a mark? Our data indicates that increasing depth increases response time linearly. The limiting factor appears to be error rate. Error rate was found to rise significantly for menus beyond the 8,2 menu. 8,2 menus were not any more unreliable than 4,2 menus. Common sense tell us that the marks required to select from a 4,2 menus are not difficult to draw. Hence we consider menu configurations which did not significantly differ in error rate from 4,2 menus to be reliable. It seems reasonable to recommend using menus of breadth four, up to depth four, and menus of breadth eight, up to depth two. 12,1 menus border on unreliability.

Off-axis analysis indicates that the source of poor performance at higher breadths and depths is due to selecting off-axis items. Thus, when designing a wide and deep menu, the frequently used items should be placed at on-axis marks. This would

allow some items to be accessed quickly and reliably with marks, despite the breadth and depth of the menu.

What is an acceptable error rate? The answer to this question depends on the consequences of an error, the cost of undoing an error or redoing the command, and the attitude of the user. For example, there is data that indicates, in certain situations, experts produce more errors than novices (Sellen, Kurtenbach, & Buxton, 1990). The experts were skilled at error recovery and thus elected to sacrifice accuracy for fast task performance. Our experiences with marking menus with six items in a real application indicate that experts perceived selection to be error-free. Other research reports that radial menus with up to eight items produce acceptable performance (Hopkins, 1991). Marking menus present a classic time versus accuracy tradeoff. If the marking error rate is too high, a user can always use the slower but more accurate method of popping up the menus to make a selection.

Marking error rates can be compared to linear menu error rates but one must be very cautious when comparing results from different experiments and different interaction techniques. Even within the same experiment, subjects may not consistently perform at the same level of accuracy, or the experimental task may artificially inflate or deflate the error rate. We can, however, make some approximate comparisons. In a study of selection performance using pop-up hierarchic linear six-item menus of depth two, Nilsen (1991) reports error rates of 2.3%. Nilsen also reports that subjects accidentally popped up the wrong submenu on their way to making a correct selection 6.3% of the time. In another study of similar pop-up linear menus, Walker, Smelcer, & Nilsen (1991) report error rates that range from 2.0% to 12.6% for subjects selecting from nine-item menus of depth 2. These error rate figures are in the range of the error rates found in our experiment for menus of up 8,2 menus. Therefore, we can conclude, with caution, that marks, within the limits discussed above, can be as accurate as selection from linear menus. It is also critical to note that this level of accuracy is not the expense of speed. For example, in this experiment selection from 8,2 menus required on average 1.5 seconds. In Nilsen's experiment, selection from six-item linear menus of depth 2 required on average 1.8 seconds (six-item menus should be faster). We found that, comparing the data from Nilsen and Walker experiments with this experiment, for equivalent menu configurations, selection from linear menus is slower than selection using marks.

<center>125</center>

Q3: Is breadth better than depth? For menu structures that resulted in acceptable performance, breadth and depth seems to be an even tradeoff in terms of response time and errors. For example, accessing 64 items using 4,3 menus, is approximately as fast as using 8,2 menus. Both have approximately equivalent error rates. Thus, within this range of menu configurations, a designer can let the semantics of menu items dictate whether menus should be narrow and deep, or wide and shallow.

Q4: Will mixing menu breadths result in poorer performance? The experiment did not show this to be true. One possible explanation is that our menu labels strongly suggested the correct angle to draw and thus eliminated confusion. A stronger test would use less suggestive labels when mixing breadths. Our results do indicate that, when there is enough familiarity with the menus, mixing breadths is not a significant problem.

Q5: Will the pen be better than the mouse for marking menu marks? Overall, the pen proved to be more suitable. However, for small menu breadths and depths, the mouse produced approximately equivalent performance. We found this extremely encouraging because it implies that marking menus are an interaction technique that not only takes advantage of the pen but also remains compatible with the mouse. Of course, it is worthwhile to note that some subjects thought their performance with the mouse was just as good as with the pen, but that the mouse required more effort to attain this level of performance.

These conclusions should be tempered by reminding the reader that this experiment simulated an expert situation (i.e., subjects were asked only to select from three different targets, thus they quickly became "expert" at those targets). We have argued that this situation is reasonably realistic. Other realistic situations, such as the performance of users on unfamiliar hierarchic marking menus with varying targets, has yet to be explored.

5.4. SUMMARY

The chapter described an experiment to test the limitations of using marks to select from hierarchic marking menus. Subjects were asked to select from marking menus using marks only. Menus were chosen such that the subject would very quickly learn and remember the mark required to perform a given selection. The breadth and depth of these menus and the input device was then systematically varied to

126

elucidate the effects of these variables. Subjects' time to perform selections and error rates were collected and analyzed. Subject's perceptions were collected using a questionnaire.

The experiment revealed that error rate was the limiting factor. Menus of breadth 4, 8 and 12 were examined. Error rate became a factor when menu breadth was eight or twelve. For these breadths of menu, error rate rose significantly when depth was greater than two. For these menu structures with acceptable error rates, there appeared to be an even depth/breadth tradeoff. When menus structures contained equivalent numbers of items, subjects showed equivalent performance on both narrow, deep menus and wide, shallow menus. It was also discovered that mixing menus of different breadths in a menu structure did not adversely affect performance. Finally, we concluded that the pen is more suitable for drawing marking menu marks than the mouse, but the difference is not large.

This chapter has answered some basic questions about the design variables of hierarchic marking menus. Specifically, how deep and wide can menu structures be yet still allow a user to perform selections using marks? The following chapter takes the answers to these questions and applies them to the design of hierarchic marking menus in a pen-based application.

Chapter 6: Generalizing the concepts of marking menus

6.1.   INTRODUCTION

This chapter reports on a design experiment which deals with applying the design principles of self-revelation, guidance, and rehearsal to interface design. Two issues are explored. First, we examine the ramifications of integrating an interaction technique that is based on these principles (marking menus) into a pen-based interface. We found that it is possible to integrate marking menus into an interface but several compromises needed to be made. Although these compromises change the original design of marking menus, we show that the resulting design still obeys our three design principles. Second, we examine how these design principles can be applied to other types of marks besides zig-zag marks. With this goal in mind, we developed an interaction technique that provides self-revelation, guidance, and rehearsal for these other types of marks. These experiences provide a better understanding of the role of marking menus in interface design and demonstrate the value of the design principles.

The test bed for this design experiment was a pen-based electronic whiteboard application called *Tivoli* (Pederson, McCall, Moran, & Halasz, 1993). 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* (see Figure 6.1) (Elrod et. al., 1992), that can be written on with an electronic pen (see Figure 6.2). Much like a whiteboard, several people can stand in front of a Liveboard and write, erase, gesture at, and discuss hand drawn items.
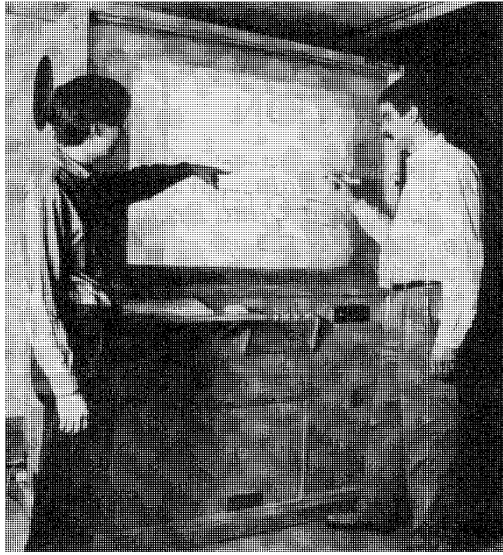
129

Figure 6.1: The Liveboard in use.

Tivoli, however, does more than just emulate marking on a whiteboard. Marks can be edited, stored, and retrieved. Marks are remembered by Tivoli in terms of *strokes*. A stroke is the path of the pen recorded from the moment the pen is pressed against the screen and moved, until it is released from the screen. A screenfull of strokes can be grouped into a "slide", and saved for retrieval later. Typical operations on strokes include moving or copying groups of strokes, changing the color or thickness of the pen tip, and undoing edit operations. Users draw "edit marks" to perform some of the editing operation described above. Figure 6.3 shows the types of marks used. Other operations are triggered using graphical buttons, dialog boxes, and menus.

One basic goal of our design study was to address the problem of operating extremely large displays. It is envisioned that someday the *Liveboard* display surface would be very large, and therefore, we wanted to address the problem of bringing the commands to the user as opposed to the user moving to the commands. Marking menus seemed suitable for this type of design since the menus can pop up at any location and the marks can be made at any location.[20] Furthermore, since

---

[20] This is not completely true. Depending on the design of the interface a user may have to be over some particular area or object on the display before a menu can be popped-up or a marking intepreted. However, the point is that pop-up menus and marks help reduce the amount of movement a user must make to invoke functions. For example, when a user wants to change pen color, traditionally one has to move from the drawing

130

Tivoli has many commands, we felt that hierarchical marking menus might allow access to many of these commands from a single location. The issue was whether or not we could integrate marking menus into the existing Tivoli interface design to solve some of these problems.

Another basic design goal was that Tivoli should be based on the unfolding interface paradigm described in Chapter 2. For example, for a novice Tivoli user the interface presents a limited set of functions—the type of functions one gets with an ordinary whiteboard. However, additional functions can be discovered and used with minimal instruction and experience. In effect, once a user has the "key" to unlock the hidden functionality, Tivoli can be unfolded and additional functions invoked. Using edit marks is a way to hide additional functions. The edit marks are not in themselves self-revealing, and therefore, this serves as a way to hide functions from a novice.

---

area to a color pallet and back. With a pop-up menu, this trip is avoided since the menu can be popped-up over some white space in the drawing area.
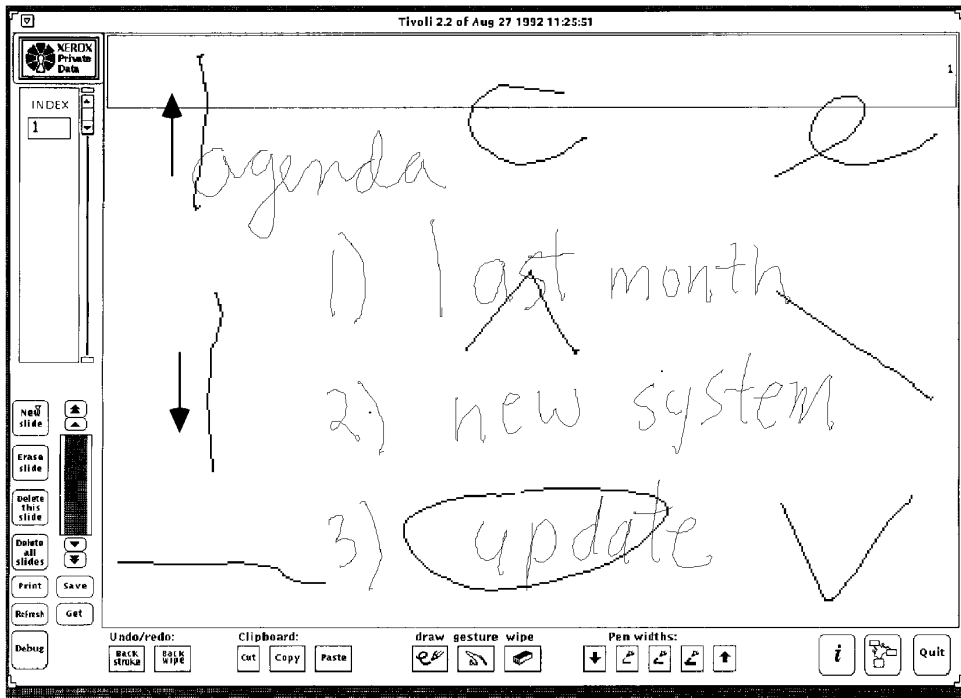
Figure 6.2: An application called Tivoli, running on Liveboard, emulates a whiteboard but also allows drawings to be edited, saved and restored.

Given these basic goals, we explored two problems. The first problem was to determine which Tivoli functions would be suitable for marking menus, and how marking menus could be integrated into the existing interface. The second problem was how to provide self-revelation, guidance, and rehearsal for the edit marks in Tivoli.

6.2. INTEGRATING MARKING MENUS INTO A PEN-BASED INTERFACE

We decided we would explore design issues by using marking menus to control pen settings in Tivoli. In Tivoli, the pen can be set to different colors and thicknesses. Originally, these settings were performed using a pallet of buttons which had an individual button for each pen thickness and color. There were several reasons why it would be advantageous to control these functions using marking menus. First, the original buttons consumed a large amount of screen space. Replacing these buttons with a marking menu would free up this screen space. Second, changing pen

132

settings was a frequent operation while drawing. Changing settings meant many trips to and from the button pallet. A marking menu could be made to pop up at the drawing location, thus avoiding trips to the button pallet. Third, no intuitive set of marks exist for controlling pen settings. Marking menus could provide a set of marks and a method for learning those marks.

6.2.1.   Adapting to drawing and editing modes

Figure 6.4 shows the marking menu we used to control pen thickness and color. The range of items is deliberately small. We felt that, in Tivoli, users need only a few different thicknesses and colors for the pen. This is like real whiteboards, where the number of markers is limited. The menu items "inc" and "dec" allow a user to increment and decrement the pen size to get custom thicknesses. The menu appears when a user presses-and-waits with the pen anywhere in the drawing area. This allows a user to change pen settings without having to move the pen from the current drawing location.
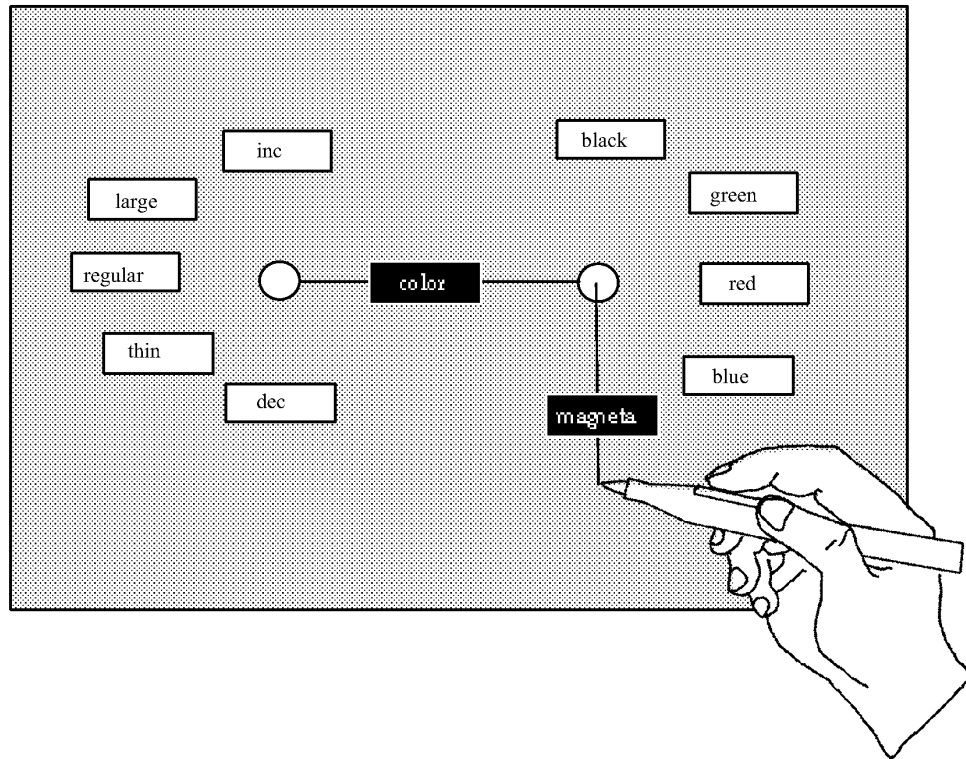
Figure 6.4: The hierarchical marking menu used to control pen settings in Tivoli. The menu can be popped up by pressing-and-waiting instead of drawing.

There is a complication with this design. Normally, a marking menu allows a user the alternative of drawing a mark to select a menu item. However, in the situation just described, Tivoli is in the "drawing mode" (i.e., all marks are interpreted as drawings, not commands). A mark is interpreted as a command in Tivoli when it is drawn while a button on the pen (the command button) is pressed. Thus, the design of the Tivoli's interface requires that menu selection marks (which are actually command marks) be performed with the command button pressed. However, this deviates from the rehearsal principle slightly: the physical action of making a selection mark is the same as selecting from the menu, but the command button must also be pressed. All the directional motions remain the same so we can be hopeful that using the menu still develops skills useful for learning and making the selection marks.

We have no empirical data to verify that, despite this deviation in rehearsal, skills developed in using the menu are still transferred to using the marks. However,

when using Tivoli ourselves, because of our experiences with the menu, we were able to recall the spatial layout of the menu, and issue marks. The role of spatial memory and physical movement memory in the transition from menus to marks is a topic for future research.

6.2.2. Avoiding ambiguity

Typically interfaces that use marks as commands identify marks by the shape of the mark or the context in which the mark is made. This discussion discriminates between marks intended for marking menu selections and other kinds of marks intended for commands. For the sake of brevity in this discussion, we will refer to these other kinds of marks as *iconic marks*, although the meanings of these marks may not be strictly based on iconic shape (see section 6.3.1 for further discussion). Also for the sake of brevity, we refer to marking menu's zig-zag marks as *menu marks*. The important point is that the potential exists for marking menu marks (menu marks) to be confused with iconic marks. Figure 6.5 shows an example of two marks for a menu structure of breadth eight and depth of two which are the same as some of the iconic marks in an early version of Tivoli.

These types of ambiguities are not peculiar to marking menu marks. Many interfaces that use marks exhibit this problem. For example, a classic problem is drawing an "O" for the letter "O" and having it confused with a small circle (where circling performs a selection). We present three strategies for overcoming this problem for marking menus, and the advantages and disadvantages of each one. We then describe how a one of these three strategies was used in Tivoli.

Avoidance

One way to avoid ambiguities between marking menu marks and iconic marks is to eliminate the ambiguous marks from the marking menu set. This can be done by avoiding the placement of menu items at locations in a menu structure that would result in ambiguous marks. These "avoided locations" can be occupied by null menu items. A mark that selects a series of null items is then considered no longer a marking menu mark, and therefore ambiguity is eliminated.

135

Marking menu marks
for breadth 8, depth 2 menu

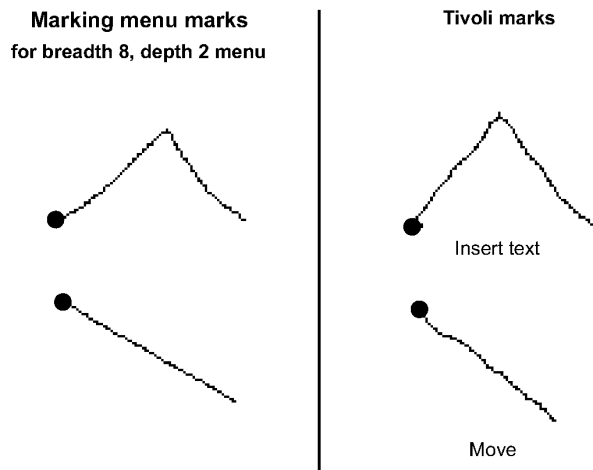**Tivoli marks**

Insert text

Move

Figure 6.5: The marks used for a marking menu may conflict with other marks. The example shows two marks used for selecting from a marking menu that can be confused with edit marks in an early version of Tivoli. A dot indicates the starting point of the mark.

One drawback to this approach is that the number of items a menu can hold is reduced and "unnatural" gaps may appear in the menus. For example, suppose a menu contains an ordered set of font sizes. If one of the menu items is not used, then a gap appears between two menu items that logically should appear adjacent to one another. This may make learning the layout of the menus more difficult.

Another drawback is that eliminating a menu item from certain location forces that the item to be placed somewhere else. Menu structures can be expanded to hold displaced items either in breadth or in depth. As shown in Chapter 5, expanding in breadth or depth slow menu selection and increases errors. Furthermore, eliminating items may result in losing on-axis items, which have been shown in Chapter 5 to enhance performance. Ultimately, rearranging menus may lead to menus that appear to be oddly structured, and this results in menus that are hard to learn, slow to use and error-prone.

These drawbacks makes avoidance a poor solution. In certain restricted cases, though, it can be a simple and easy solution to implement. For example, suppose the only conflicting mark is a horizontal stroke which is to the right, and the marking menu only needs to contain six items. The simple solution is to use an

136

eight-item menu and the make the "right stroke" menu item and some other menu item null items, and populate the remaining menu items with the six commands. A variation on this strategy is to change the iconic marks. This, of course, avoids the problems with modifying a marking menu as described earlier, but in certain situations may cause confusion for a user when obvious or common marks are replaced by non-obvious, uncommon iconic marks.

Different context

Another design alternative is to allow iconic marks and marking menu marks of the same shape to coexist but determine their meaning by the context in which they are drawn. Two dimensions in which the context can vary are time (i.e., when the system is in a certain mode a mark has a certain meaning), and by space (i.e., a mark's meaning varies depending on the location at which it is drawn).

Distinguishing the meaning of a mark by the context of time leads to moded interfaces. An interface where a user must enter a "marking menu mode" to issue a marking menu mark seems to defeat the purpose of making a mark—a fast way to invoke a particular command. However, if the cost of switching modes is very low and properly designed (Sellen, Kurtenbach, & Buxton, 1992), this can be an effective technique. An example of low cost mode switching is a dedicated pop-up menu button on the mouse which is found in many windowing systems such as *X11* (X11, 1988) and *Open Look* (Hoeber, 1988). After developing the habit of holding down the button to pop-up and maintain a menu, a user no longer perceives using a menu as a mode. One can imagine such a similar design for marking menus where a user presses down a button on the pen or mouse to indicate to the system that the mark is intended for the marking menu. The obvious disadvantage to this scheme is that a hardware button must be dedicated strictly to a menu. Many pen-based system pens do not have buttons, or the buttons have already been assigned other functions. For example, in Tivoli the two buttons on the pen were already used for other functions. The first button is used to distinguish between drawing mode and command (edit mark) mode. The second button is used to control whether the pen is in drawing mode or erasing mode.

Another type of context that can be used to distinguish the meaning of a mark is location. For example, a stroke through a word may mean "delete the word" while the same stroke starting on a graphic may mean "move the graphic". This type of

137

scheme works well with object oriented direct manipulation systems, where the combination of an object and a mark can be used to distinguish a mark's meaning. Of course, distinguishing meaning by location will not work if the same location must accept two identically shaped marks.

Marking menus work very well in identification by location situations. For example, on a different project, we found an effective interaction technique can be created by embedding a marking menu in an ordinary graphical button. In effect, this extends the functionality of the button. Along these lines, we developed a system called *HyperMark* which allows marks to be used in Apple's Hypercard (Apple 1992). For example, if HyperMarks are added to a button, not only does a button react to a mouse press, but marks can also be drawn on the button which trigger other actions. This results in the interface having fewer buttons and faster interactions in some cases. In effect, HyperMarks are similar to pop-up menus where additional functions are hidden under a button until popped up. However, with HyperMarks, a user does not have to wait for a menu to pop up, visually search the menu and point to an item. Instead, a mark triggers the item directly. Our intention was to permit ordinary Hypercard users or programmers to incorporate marks into their own Hypercard stacks.

With HyperMark, different buttons accept the same mark but the interpretation of the mark is different. Figure 6.6 shows an example of different locations having different menus but reusing the same set of marks. The meaning of the marks is disambiguated by the location of the mark. We feel this is a reasonable design as long as the common commands (scroll up and scroll down, for example) are kept consistent from button to button.

The disadvantages of discriminating by location are, first, it does not eliminate the problem if the same location accepts two ambiguous commands and second, it consumes screen space. Consumption of screen space results in situations where the desired location is not displayed on the screen and must be acquired by the user. This can slow interactions and defeat the purpose of using marks.
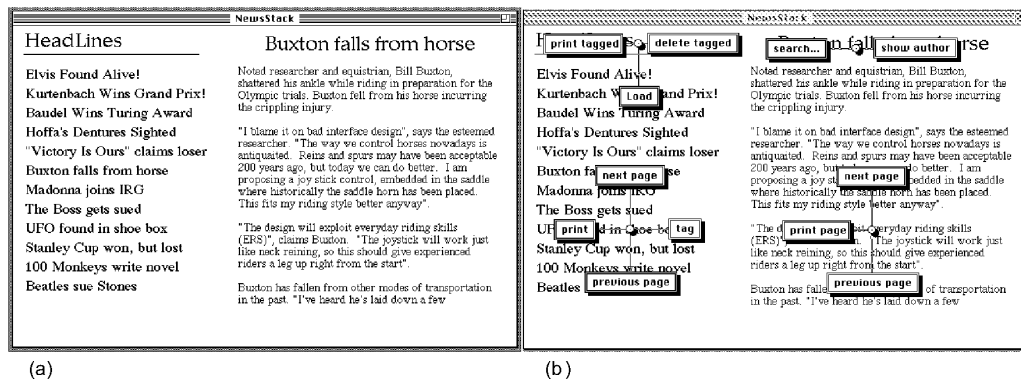
138

Figure 6.6: A simple news reader program in Hypercard that is controlled by marking menus. (a) shows the four major area of the screen: "Headlines", a list of articles, the title of the current article, and text of the article. (b) shows the marking menus associated with each of these areas. When marks are used to select from the menus the context (the location) of the mark contributes to its meaning.

Distinguishing tokens

Distinguishing marks by tokens involves augmenting a mark with some characteristic that disambiguates it. Augmentation can be of several forms. The shape of marks can be augmented. Alternatively, the dynamics of drawing the mark can be used to augment a mark.

Figure 6.7 shows how an augmenting "dot" at the start of a marking menu mark is used to indicate the mark is intended for a marking menu. An augmenting token, however, does not have to be at the start of the mark. The token could appear as a prefix to the mark, within the mark or as a suffix to the mark. However, if the mark is not distinguished from the start, then mark-confirmation may lead to ambiguities, since the system may identify the partially completed mark as both the start of a marking menu mark and an iconic mark.

139

Figure 6.7: Two marking menu marks that are augmented by a "dot" to distinguish them from other types of marks in an interface.

There are many alternatives to "dot". Any sort of token that guarantees distinction could be used. In practice, we found "dot" easy to draw and easily and reliably recognized by the system.[21] We also found that one could make an analogy between it and press-and-wait. In Tivoli, pressing-and-waiting in drawing mode popped up a marking menu to change pen settings. "Dot" could be thought of as a mark in command mode that mimicked press-and-wait, and allowed access to the pen setting menu.

Another way of distinguishing marks is by dynamics. For example, in some systems the speed at which a mark is drawn determines its meaning. For example, a slow up-stroke may mean "next page", while a quick up-stroke (a "flick-up") may mean "go to the end of the document" (Go, 1991). In Tivoli, we experimented with dynamic schemes and found several problems. First, flicks are not consistently recognized because the speed of a flick varied with direction and the user's dexterity. Also, quick movements sometimes caused the pen to skip off the display surface before the speed of a flick could be attained. Flicking was not very reliable because of these problems. We also experimented with prefix flicks and suffix flicks. Prefix flicks made drawing the remaining mark too hard: slowing the pen down after drawing the flick to draw the rest of the mark, was difficult. Alternatively, drawing the entire mark at flick speed was too hard. Suffix flicks were more reliable: we could safely draw the first part of the mark then add a "flick flourish" on the end of the mark to indicate it as a marking menu mark.

---

[21] We occasionally operated Tivoli with a mouse, although it is intended to be operated with a pen. In this case we found a "dot" very difficult to draw. Thus we would not recommend the use of the "dot" for mouse driven system that use markings.

140

Recognizing flicks was further complicated by limitations in the input event software. On occasion, input events are buffered. Time stamping of input events occurs after events are read from the input buffers and therefore, at times, these buffering delays confuse the flick recognition process. This problem could be overcome by immediately time stamping all events. Nevertheless, this indicates that tracking dynamics place special demands on input software.

Even if flicks could be made reliable they still present a problem: how can flicks be demonstrated to a user? The "dot" is easy to learn because a user can simply be told: "make a dot, about this big". Flicks on the other hand are dynamic in nature and are best learned by demonstration and practice. Section 6.3.2 discusses issues concerning self-revelation of mark dynamics.

To summarize, we have presented three strategies to avoid ambiguity between menu marks and iconic marks: avoidance, different context, and distinguishing tokens. Based on the various advantages and disadvantages each strategy just discussed, we elected to use a distinguishing tokens strategy in Tivoli. Specifically we used the "dot" prefix mark shown in Figure 6.7. Section 6.4 discusses our experiences with this strategy.

6.2.3. Dealing with screen limits

One problem that can occur in a pop-up menu system is that, when a menu is displayed near the edge of a screen, some portion of the menu may be clipped-off. This may make it impossible to see or select some items. We refer to this as the *screen limit* problem. Marking menus suffer from this problem because they use pop-up menus.

One possible solution to the screen limit problem is not to allow menus to be displayed too close to the edge of the screen. This implies placing menu "pop-up spots" some safe distance away from the edge of the screen. While this is a workable solution, it is not practical when menu hierarchies are deep, since pop-up spots may have to be located a large distance from the edge of the screen to keep the submenus from hitting the edge of the screen. Furthermore, it seems to be an unreasonable constraint given popular interface design. For example, most drawing programs have scrollable windows, and a user is allowed to scroll a window till menu pop-up spots are close to the edge of the screen.

Another solution to the screen limit problem is *constraining.*, Most pop-up menu systems constrain menus to display entirely on the screen, even if the location from which the menu was invoked would cause some portion of it to be clipped-off. For example, the menus in Open Look use this solution (Hoeber, 1988). Constraining, however, causes problems when hierarchic menus are used. In this case, accessing a series of menus causes each menu to hit the edge of the screen. We refer to this problem as *crowding*. When crowding occurs, users end up making a series of selections from menus that are against the screen edge and this can sometimes make menu selection slow and error-prone.

Hopkins (1991) uses a constraining solution for radial menus. Since marking menus use radial menus, it is worthwhile to consider this solution. With Hopkins' radial menus (or pie menus), normally, a pie menu pops up centered around the cursor location. However, when the cursor is close to the edge of the screen, this results in some portion of the menu being clipped-off, To overcome this problem the menu is displayed not centered around the cursor, but shifted over so it is completely displayed. The cursor is then reset by the system to the center of the menu (this is referred to as "warping" the cursor). At this point, the user can make a selection in the usual way.

Problems occur with Hopkins' solution when the input device is an absolute device like the pen, and this makes it unsuitable for marking menus in Tivoli. The problem is that the system cannot change the location of pen (given the constraint that the cursor always appears under the tip of the pen). An example demonstrates this. Suppose a radial menu is popped up too close to the edge of the screen. If the menu is constrained to display completely on the screen, the pen tip is no longer in the center of the menu. The pen tip generally ends up located in one of the menu items. This immediately highlights the item. If the highlighted menu has a submenu, this menu would then be displayed. Thus, a user inadvertently descends the menu hierarchy. Even if the menu item has no submenu the user would still have to move the pen out of the menu item if the menu item was not the desired one.

We propose the following solution which permits marking menu selections near the edge of the screen when using a pen. When the pen is pressed close to the edge of the screen, the marking menu appears centered around the pen tip cursor with some portion of it clipped-off. If the clipped-off portion is large enough to obscure some menu items, another special menu item (referred to as the "pull-out" menu item)

142

appears on the screen (see Figure 6.8). At this point a user can select the visible menu items in the normal fashion. However, if the user moves the cursor to the pull-out menu item, the menu is redisplayed centered at the location of the pull-out item. The pull-out menu item is located far enough away from the edge of the screen so that the menu is completely visible when redisplayed. At this point the pen is located in the center of the menu and all items are accessible. This same scheme works with hierarchic menus. Every time a submenu hits the edge of the screen, a pull-out item is displayed.

143
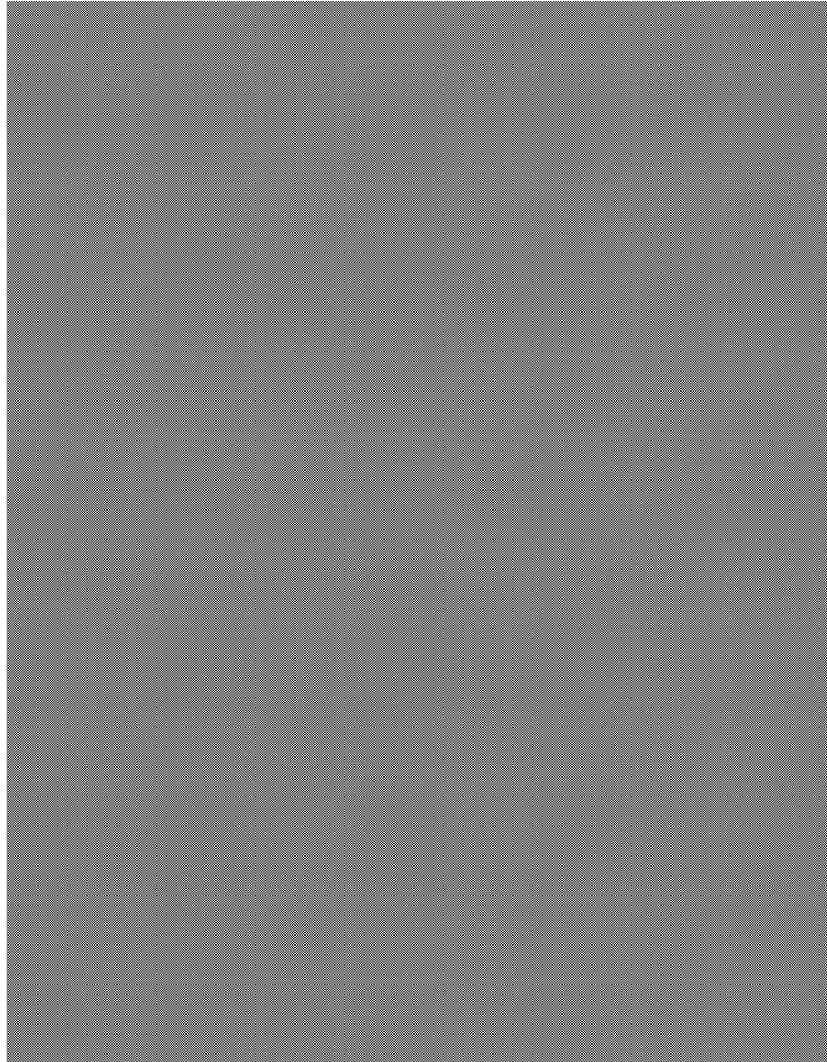
Figure 6.8: A "pull-out" menu item allows a user to access menu items that would be clipped-off by the edge of the screen. In (a) a user has displayed a marking menu but a portion of it is clipped-off by the edge of the screen. Because of this, a pull-out item appears (the gray circle). In (b) when the user drags over to the pull-out item, the menu is redisplayed so all items can be accessed.

Marks also have a screen limit problem. If one starts a mark too close to the edge of the screen one may run into the edge. As with menu mode, the input device used makes an important difference in a solution to the problem.

If a relative input device like the mouse is used, it is possible for users to draw marks "beyond" the edge of the screen. Hopkins (1991) has proposed a solution that

144

is suitable for marks. Hopkins' pie menus use a technique called mousing-ahead which is similar to marking but the path of the cursor leaves no ink-trail (see Section 2.3.1 for a complete explanation of mousing-ahead). Mousing-ahead is possible even when the cursor hits the edge of the screen. Although the cursor is constrained to the area of the screen, mouse movement after the cursor hits the edge of the screen is still tracked. Thus, a user can mouse-ahead beyond the edge of the screen. Applying this solution to marks, a user could draw marks beyond the edge of the screen, although some portion of the mark would not be visible. This solution is important because it preserves the principle of rehearsal. The movement to select from the menu must be the same as movement to make a mark and this happens even when menus and marks hit the edge of screen.

If the input device is a pen, drawing a mark close to the edge of the screen behaves logically: if the mark does not hit the edge of the screen, it can be performed as usual; if the mark does hit the edge of the screen, a user cannot physically draw it. This behavior mimics the way pen and paper works—if one is too close to the edge of the page one cannot draw certain marks.

We still need to, however, be able to apply marks to objects that are near the edge of screen. To do this we mimic pen and paper traditions. Generally, when something is too close to the edge the page to fit, a line is drawn from the object, out to some clear space and then an annotation is made. We propose a similar design. Suppose an object is too close to the edge of the screen for a certain mark to be made. A user can draw a line, out to some clear space on the screen, then make a "pull-out" mark, followed by the desired mark. Figure 6.9 shows this.

6.3. APPLYING THE PRINCIPLES TO ICONIC MARKINGS

Marking menus provide self-revelation, guidance, and rehearsal for "zig-zag" types of marks, specifically, the type of marks that are the byproducts of selecting from radial menus. Can a similar mechanism be provided for iconic marks? As a design experiment we decided to see if we could design mechanisms similar to marking menus but for the edit marks in Tivoli. Thus we attempted to design ways to self-reveal these marks, guide a user in making them, and have this be a rehearsal which builds skills for expert behavior.
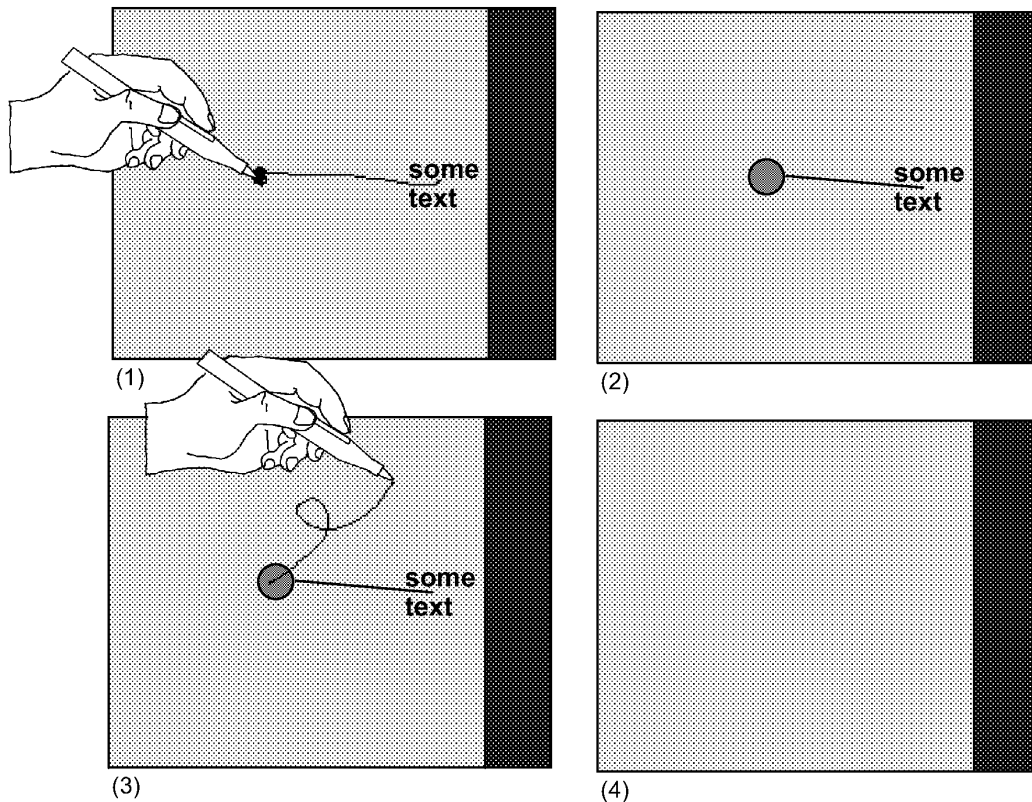
145

Figure 6.9: Using a "pull-out" mark to apply a mark to an object close to the screen edge. In (1), the pull-out mark is drawn (a line followed by a dot). In (2), the system has turned the mark into a pull-out object. A mark is then drawn in the pull-out object, in (3). In (4), the mark is applied to the object that is "pulled out", and it is deleted.

Another design goal was ease of programming. One of the attractions of marking menus is that an interface programmer can implement interactions which provide self-revelation, guidance, and rehearsal with something as simple as a pop-up menu subroutine call. We wanted a mechanism for iconic marks that was just as convenient to program. The idea was to avoid creating custom code to self-reveal each different type of mark.

6.3.1. Problems with the marking menu approach

Overlap

Suppose we strictly applied the marking menu design to the marks shown in Figure 6.3. In other words, display all the possible marks a user could make starting from a

146

certain location. Figure 6.10 shows the result of this approach. Marks overlap and can cause confusion. Part of the problem is that iconic marks are not suitable for displaying in this manner. Menu marks, however, are suitable because of their directional nature. Another problem in the example is that each entire mark is displayed. If all the marks of a hierarchical marking menu were displayed, this too, would result in overlap.
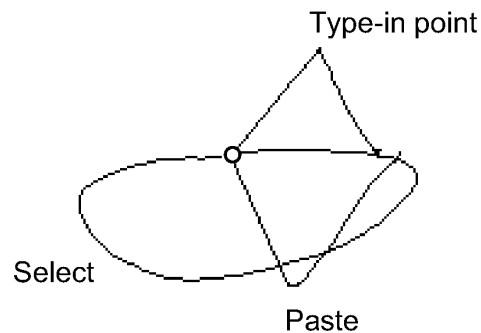

Figure 6.10: Overlap causes confusion when using the marking menu approach to self-reveal other types of marks. Here we display the commands available when starting a mark from a clear spot in the drawing region of Tivoli.

Not enough information

A display like Figure 6.10 gives little contextual information. For example, the important thing about the "Select" mark is that it should encircle objects and the shape of the circle can vary. This type of information is not shown in Figure 6.10.

The meaning of several edit marks in Tivoli is determined not only by the shape of the mark but also by the context in which the mark 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 edit marks a user can make over what objects. Informally, we wanted a user to be able to answer the question: "what marks 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 marks in Tivoli.

147

For mark sets in general, besides Tivoli's iconic mark set and the marking menu mark set, the following characteristics may contribute to a mark's meaning and this type of information therefore needs to be self-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 mark affects its meaning. For example a up-stroke means "scroll up" while a down-stroke means "scroll down". The shape of the mark is basically the same but the direction or orientation of the mark has meaning.

Location of features: The location of particular features of a mark can affect its meaning. For example, the summit of the "Type-in" point mark shown in Figure 6.10, determines the exact placement of the text cursor.

Dynamics of drawing: How a mark 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".

6.3.2.   Solutions

Crib-sheets

Interactive crib-sheets self-reveal marks without the overlap problem. When the user requires help, a crib-sheet can be popped up which shows the available marks and what they mean. The user can then dismiss the crib-sheet (or "pin" it down on the side) and make a mark. In Chapter 1, two systems that use mechanisms similar to this were described. Crib-sheets can be as succinct as a simple list of named marks or as elaborate as multi-page explanations of the marks in great detail. Thus a crib-sheet could contain complete information on all the characteristics of a mark. However, since crib-sheets are for reminding and guidance, they are usually succinct.

Figure 6.11 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 marks can be applied to a certain object or location on the screen, one presses-and-waits over the object for more information, rather than marking. At this point, rather than a menu popping up as in the marking menu case, a crib-sheet is displayed. The crib-

148

sheet displays the names of the functions that are applicable to the object or location, and example marks. If this is enough information, a user can draw one of the marks in the crib-sheet (or take any other action) the crib-sheet automatically disappears. If the pen is released without drawing mark, the crib-sheet remains displayed until the next occurrence of a pen press followed by a pen release or a press-and-wait event.
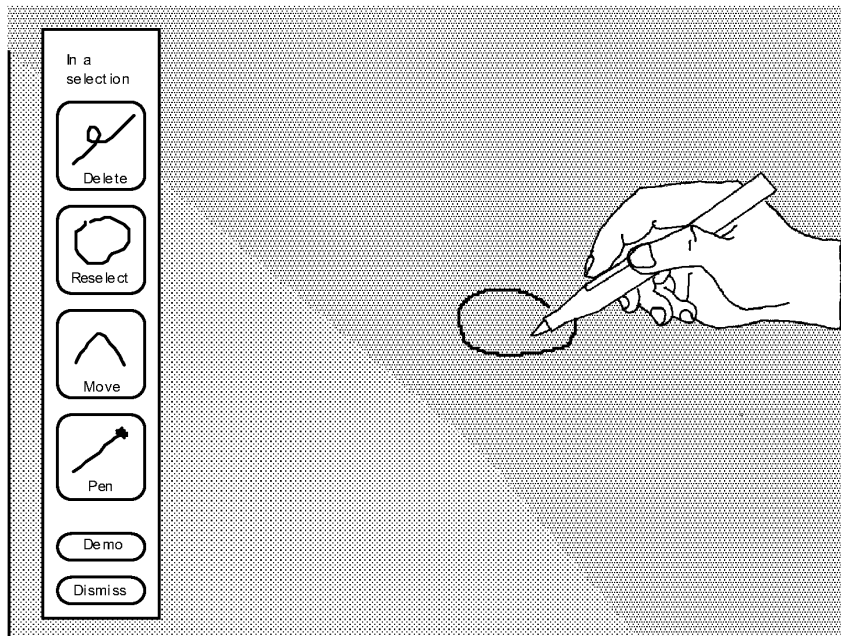

Figure 6.11: Self-revealing iconic marks in Tivoli: The user has selected the word "Tea" by circling it. To reveal what functions can be applied to the selection, the user presses-and-waits within the selection loop. A crib-sheet pops up indicating the context ("In a selection") and the available functions and their associated marks.

This design has several important features. 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 mark. Second, the significance of the location of the pen tip is displayed at the top of the crib-sheet (i.e., in Figure 6.11 "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 self-revelation, guidance, and rehearsal. The crib-sheet provides self-revelation, and a user can use the examples as guidance when drawing a mark. Rehearsal is enforced because a user must draw a mark to

149

invoke a command. For example, a user cannot press the delete button on the crib-sheet to perform a deletion. The user must draw a delete mark to perform a deletion.

Animated, annotated demonstrations

While the crib-sheet does self-reveal contextual information about marks, it still lacks certain types of information. For example, one static example of a mark relays little information about variations and features of a mark. It has been shown that people need good examples to help visualize procedures (Lieberman, 1987). Ideally a demonstration of the mark 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 marks are made by animated examples.

Demonstrations can be provided through animation. Baecker and Small have described how animation can assist a user, and how the animation of icons can be effective (Baecker & Small, 1990; Baecker, Small, & Mander, 1991). The idea of animated help is not new. Cullingford (1982) used "precanned" graphical animation coupled to natural language contextual messages to provide help. Feiner (1985) used graphical explanations to illustrate the problem solving process of real world physical actions. Feiner's system, however, was not sensitive to the user's current context. A research system, called *Cartoonist*, which automatically generates context sensitive animated help for direct manipulation interfaces, has been developed (Sukaviriya & Foley, 1990; Sukaviriya, 1988). The major difference between Cartoonist and the system we are about to describe is that Cartoonist is designed for direct manipulation interfaces, not mark-based interfaces. As we shall see, an animation of drawing a mark must have special features to make it meaningful and helpful. Specifically, in our system, the animation of a mark is annotated with text for explanation. Cartoonist does not support annotations. Furthermore, Cartoonist relies on an extensive knowledge base to describe the application and interface. The system we describe has a vastly simpler implementation which is compatible with existing user interface architectures.

Crib-sheets could be animated. A crib-sheet could show how to draw a mark, variations on a mark, and the various features of a mark. This certainly would help a user understand how a mark should be drawn. However, crib-sheets illustrate

150

marks outside of the context of the material that the user is working on, and this can make it difficult to see how the mark applies to the context. Marking menus, on the other hand, have the advantage of showing the available marks directly on top the object being worked on,.

To solve these problems we extended the function of the crib-sheet by adding animations of marks which take place in context. If the crib-sheet does not provide sufficient information, a demonstration of a mark can be triggered by pressing the "demo" button on the crib-sheet. The demonstration of the mark begins at the location originally pressed. The demonstration is an animation of the drawing of the mark which is accompanied by text describing the special features of the mark (see Figure 6.12).

There are several important aspects to this design:

• Marks are shown in context. The animation of the mark is full size, and emanates from the exact location originally pressed on by the user. A user can trace the animated mark to invoke the command.

• Variations in marks can be demonstrated by multiple animations. There is usually a variety of ways to draw mark. For example, a pigtail, signifying deletion, may be drawn in any direction, clockwise or counterclockwise, big or little. To prevent users taking a single animated example too literally, we show variations by animating multiple examples of mark. Usually, two examples seems to be enough.

• Information about features is provided by annotations. Not only is the drawing of a mark animated but the animation is annotated with text to explain features or semantics of marks (e.g., in Figure 6.12 "A pigtail deletes the *selected objects*."). In addition, features of the application can be displayed. For example, in Tivoli scrolling marks can only be drawn in the margins of the drawing area, but the borders of margins are not visible.[22] In situations like this, the animation can display these features to clarify matters. Annotations appear in sequence during the

---

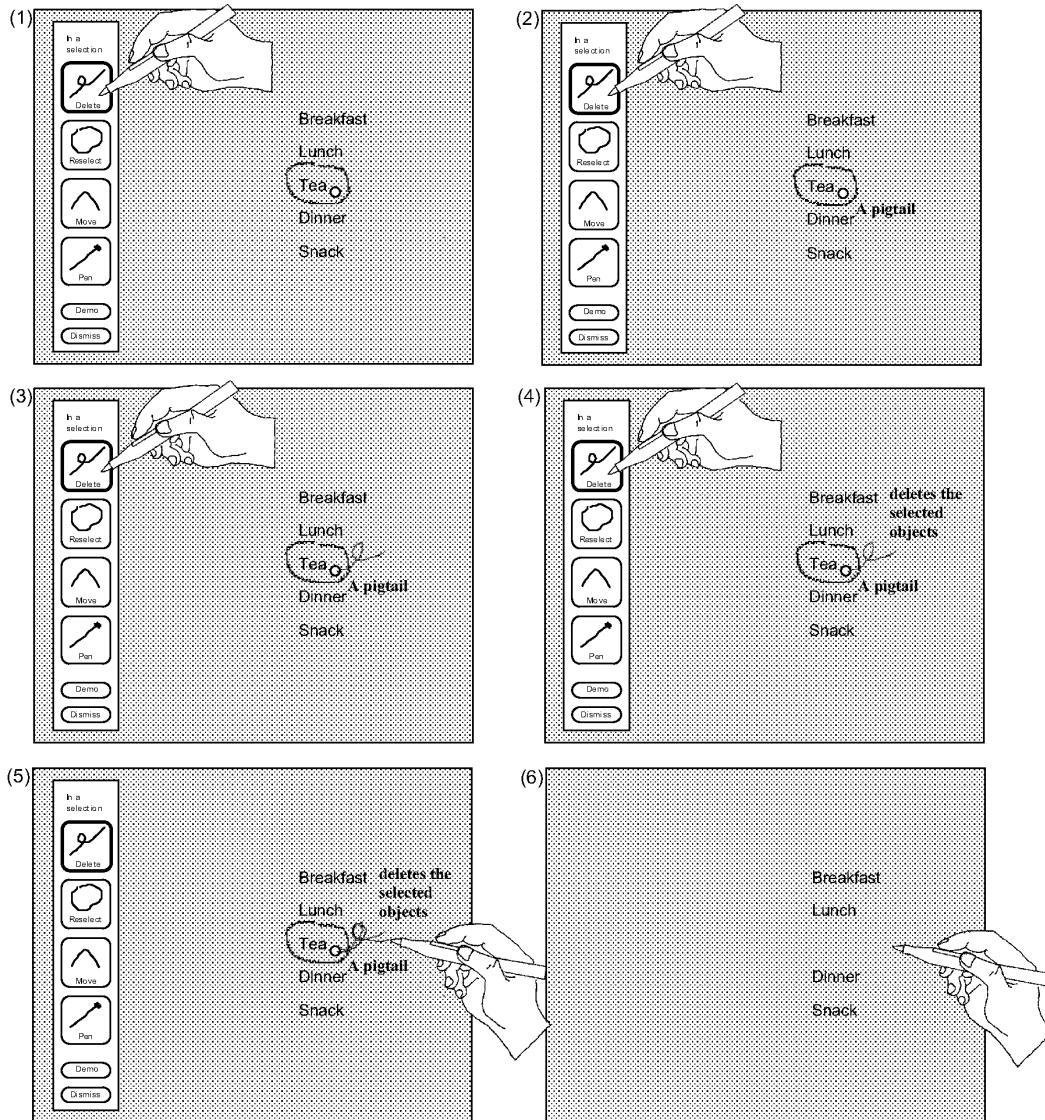[22] This was done to keep the drawing area uncluttered.

Figure 6.12: 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 mark with text annotation to explain its features. This is shown in (2), (3) and (4). In (5), the user traces along the example mark to invoke the function. When the pen is lifted, the action for the mark is carried out, and the crib-sheet and animation disappear (shown in (6)).

152

mark's animation, and they are timed to remain on the display long enough for the user to be able 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 marks can be started or stopped. Pressing "Demo All" causes the system to cycle through all the animations. Pressing the "Dismiss" button stops the animation and removes the crib-sheet. As in the case of the crib-sheet by itself, the moment a user completes a mark, the crib-sheet is removed and the animation terminates.[23]

• The user is not required to make a mark from the crib-sheet. The user is free to perform any mark at any location on the screen while the animation is running. As before, the moment the user completes a mark, the animation and crib-sheet are removed. The user can also choose to not draw a mark by tapping the pen against the screen. This also removes the animation and crib-sheet.

Architecture

A goal for our crib-sheet/animation design was that it be easy for an interface programmer to use. We designed the software architecture with this in mind. To describe the characteristics of this architecture, we will describe an interactive computer system as consisting of two parts, an application module and animator module. The application allows the user to interact with a particular domain of materials by means of marks (i.e., Tivoli is the application and the materials are free-hand drawings). The animator is called by the application to show the marks to the user. The animator is generic — it can be made to work with different applications.

The design of the animator raises many specific design problems. We describe the animator by laying out the problems and describing how they are addressed.

How does the animator get invoked? This is the job of the application. As with a marking menu, the user deliberately presses-and-waits while the command button pressed. The application detects this action and then calls the animator.

_____

[23] The animation actually freezes when a user begins drawing a marking so a user can trace the animated mark. The animation is removed from the screen when the user finishes drawing the mark and raises the pen.

153

How does the animator know which marks to animate? In order to make an application work with the animator, an application-specific Mark Animation Database (MAD) must exist. The MAD contains descriptions of examples of marks grouped by application context. When the user presses-and-waits, the application calls the animator with a description of the current context. The animator can then select the marks to be animated based on context.

How are marks and contextual features animated? In order to understand how marks are animated it is convenient to first understand the structure of MAD. Figure 6.13 shows an example of the structure of MAD. MAD consists of annotated examples of marks which are grouped by context. When the application calls the animator with a context, the examples corresponding to the context are retrieved from MAD. When a user requests a demonstration, the animator animates these examples. A mark is a sequence of x and y coordinates which is animated by incrementally displaying the mark. The marks that appear in MAD were originally drawn by hand. When animating a mark the animator uses the same drawing dynamics as the original hand-drawing (a technique developed by Baecker (1969)). In this way, dynamics of drawing can be revealed and the speed of an animation can be controlled by the constructor of the database. Annotations are labeled by where and when they should occur in the animation cycle (e.g., "start" and "end"). 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.

How are variations shown? Variations are shown by animating another example of a mark. A mark in MAD can have more than one example. If an extra example is tagged as "variation", it is then included in the animation along with the original example.

How is the crib-sheet constructed? When the animator retrieves the examples from MAD, labels for the crib-sheet buttons are extracted, and example marks are shrunk down to be displayed in the buttons. We found it convenient to designate certain example marks for shrinking. Therefore, a function in MAD can contain an extra

154

example mark that is tagged for use as an "icon" in the crib-sheet. If no "icon" example is found, the animator shrinks the first example mark it finds.

How are application features animated? Like text annotations, application features appear in MAD. If during an animation an application feature needs to be displayed, the animator makes a call-back to the application. For example, the call-back may ask the application to display the margin boundaries of the drawing area. Therefore, a call-back protocol must exist between the application and animator.
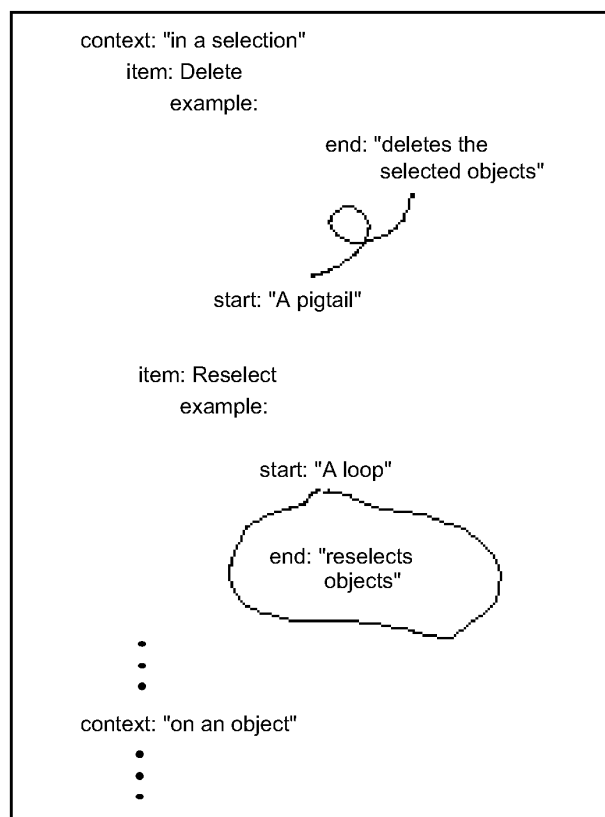

Figure 6.13: An example of the structure of the Mark Animation Database (MAD). Annotated examples of marks used for the crib-sheet and animations are grouped by context and function.

How are marks animated in constrained spaces? Assume that a user invokes the animator near the bottom of the drawing area, and that one of the possible marks at that point is a pigtail. At the bottom of the drawing area, there is no room to draw a

155

pigtail downwards, but there is room to draw it upwards. Thus, the animator should show only pigtails that fit in this place. The solution to this problem lies in the fact that MAD contains multiple examples of marks. When the animator retrieves examples from MAD it looks for examples that will fit in the space it is working with. Thus, MAD should be set up with many examples of each mark, 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 pigtail here"). This tends to only happen when there is not enough room for a user to actually draw the mark.

How is MAD constructed? MAD is constructed by drawing the examples in the form shown in Figure 6.13 and then copying these examples into MAD. For Tivoli, we constructed the examples by drawing them in Tivoli. Thus we could easily design examples that fit in constrained spaces in Tivoli by 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.

More sophisticated features

The design for the crib-sheet/animator and MAD previously described has been implemented. Section 6.4 describes experiences using it. We now discuss future designs which are currently not implemented.

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 application objects underlying the animation.

A more advanced version that we have not implemented extends the notion of parameterized marks to allow them to utilize application objects in the current working context. For example, assume we have a mark to move a list item. There would be two typed parameters to this mark: the list item and the location to which it is moved. In Tivoli, the list item would be a set of strokes between two "blue

lines" (like the blue lines on lined paper), and the location would be a blue line between two other list items. When the application calls the animator and tells it to animate a move-list-item mark, it would have to also give the animator some actual items and locations in the current context. The animator would then deform a move-list-item mark to fit the items and locations. Thus, the user would see a real example in the current context.

Having examples that manipulate the objects in the current context requires a much more sophisticated architecture for the animator. The animator must be able to manipulate objects in the application interface, and therefore a protocol that allows this must exist. Essentially, the distinction between the application and the animator becomes blurred in this more sophisticated scheme: the animator needs to know how to manipulate the application in the same way a user does. It must be able to identify objects and locations, construct marks and apply those marks. In addition, it needs to annotate the examples in a meaningful way. All these features require that examples in MAD be parameterizable. The design of this architecture is future research. 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 interfaces (Sukaviriya & Foley, 1990; Sukaviriya, 1988).

Integrating menu marks

As described earlier, menu marks in Tivoli are treated in the same manner as iconic marks. Specifically, menu marks will be interpreted as commands if drawn in command mode (i.e., drawn with the command button pressed down). The crib-sheet/animator provides self-revelation for all marks available in this mode including menu marks.

It would be impractical to include in the crib-sheet and animations all the marks used to access the pen settings menu. The menu is a much better mechanism for revealing this information, but is available only in drawing mode. Therefore, the crib-sheet/animator refers the user to the marking menu that is available in drawing mode. The animation of this is shown in Figure 6.14. The animation shows how to draw the dot required for a menu mark to be distinguished from other marks, and shows one example pen setting. The animation then displays a message for the user to see the marking menu available in drawing mode for more information. In this way an information link exists between the crib-sheet/animator and the marking

menu. Hence the crib-sheet provides self-revelation for the menu marks by referring the user to the marking menu.

A dot-stroke
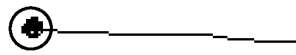
changes the pen
color or thickness,
press and hold pen
for more info

Figure 6.14: To self-reveal menu marks, the animator shows one example then refers the user to pop up the marking menu itself for more information. This avoids the problem of explaining and animating the many marks used for the marking menu.

6.4. USAGE EXPERIENCES

A large portion of the design described in this chapter has been implemented. The crib-sheet, animator, and MAD have been implemented, although the parameterized version of the animator was not implemented. Tivoli currently supports animations with multiple examples for every mark it uses. As Tivoli evolves, we expect the mark set to change. This can be supported by simply modifying MAD. The pen setting menu and marks were completely implemented. The "pull-out" menu item has yet to be implemented.

Future research will include formal user tests 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 user 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 is to evaluate some of these details and refine the content of the animations. As Baecker, Small, & Mander (1991) point out, animations require significant development and refinement. Fortunately, our design makes this easier than a frame by frame process.

The design has been used informally by several researchers at Xerox PARC. Users appeared to be quite successful at using the marking menu, once press-and-wait was

understood. Users were also successful at selection using a mark but found recognition unreliable. We traced this unreliability to incorrectly drawn "dots" at the start of marks. We found the source was not that a user failed to draw the "dot", but that the system occasionally did not start tracking the pen till after the "dot" was drawn. This implies that the pen tracking hardware and software needed improvement.

Another problem revealed through informal use was the "right-handedness" of the marking menu. Depending on a user's handedness, some portion of the screen is occluded from view when one's arm is holding the pen against the screen. When using the marking menu, left handed users found some menu items occluded from view (they had to look "under" their arms). This implied that, like most pen-based systems, marking menus must be configurable for handedness.

Users also experimented with using the crib-sheet/animator. 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 marks. 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 mark without the command button being pressed. This would result in the mark being drawn but not interpreted (i.e., the mark as drawn in drawing mode, not in command mode). 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.

6.5.   SUMMARY

In the beginning of this chapter we set out to integrate hierarchical marking menus into a pen-based application, and provide self-revelation, guidance, and rehearsal

for iconic marks. A design was developed and implemented to satisfy these goals. The design gives rise to many issues and conclusions:

The integration of marking menus into Tivoli reflects the situation with many applications today. Tivoli had an interface prior to our design experiment. Thus we were faced with the task of integrating marking menus with other interaction techniques. The main effect of this was that our design of marking menus had to change, not the existing interface components. This was an excellent test of the resiliency of the marking menus paradigm.

Marking menus had to be integrated with a range of interaction techniques. The interface to Tivoli not only contains edit marks but also free-hand drawing, buttons, dialog boxes, pop-up menus, mode buttons and a windowing system. Thus it was a challenge to find a spot where marking menus could fit in and be effective. The exploration also reminded us that interaction techniques cannot be added to an interface design without considering the other interaction techniques that surround it.

There were many other situations where we could have experimented with marking menus. One goal in redesigning the Tivoli interface was to reduce the number of buttons on the screen. Consolidating many buttons into a marking menu, hence, removing them from the screen, would have accomplished this. Also, using marking menus to issue commands to Tivoli objects such as list-items would have been another effective use. Time constrained us to only explore one particular usage. We thought using a marking menu to control pen settings would elucidate many design issues, since the menu marks would have to be used in the same mode as the edit marks. Nevertheless, this simple implementation gave rise to many design issues which one would encounter in a larger scale integration.

This design exploration also revealed issues concerning using marking menus in mark-based interfaces. Figure 6.15 summarizes the major design problems and the solutions we developed. Specifically, ambiguities can develop between menu marks and iconic marks. We proposed three solutions: avoidance, different context, and distinguishing token. We elected to used a distinguishing token strategy, given the way marking menus were being used in Tivoli. The other strategies, however, can be useful in other situations. Also this design exploration allowed us to use marking

160

Problem	Solution
Ambiguity between iconic and menu marks.	Draw a distinguishing token (a "dot") at the start of an menu mark.
Menu items clipped-off near edge of screen.	Use pull-out menu item.
Object too close to edge of screen to mark.	Use pull-out mark.
Need self-revelation for iconic marks.	Use crib-sheet/animator.
Provide guidance for iconic marks.	Draw a mark based on crib-sheet example or... Trace a mark over an example displayed by the animator.
Ensure rehearsal of iconic marks.	A mark is the only way to issue a command.
Crib-sheet/animator should be easy to program and work at any screen location.	The programmer generates multiple examples in MAD.
Getting information on marking menus marks from the crib-sheet/animator.	A crib-sheet/animator item refers user to the marking menu.

Figure 6.15: Major design problems encountered integrating marking menus into Tivoli and the solutions developed.

menus with a pen. This uncovered issues and led to developments concerning screen limits and drawing dynamics.

The crib-sheet/animator is designed to support the principles of self-revelation, guidance and rehearsal. These mechanisms do not appear and behave exactly like marking menus, and we have shown why this must be so, but we feel that the design supplies the same type of information to the user and promotes the same type of behavior.

Designing a mechanism to self-reveal iconic marks brings to light many issues concerning the self-revelation of marks. First, revelation can occur at various levels of detail. The crib-sheet is the first level: a quick glance at the icon for the mark may be sufficient for the user. An animation is the second level: it requires more time

161

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 marks themselves are self-revealed. For some marks, it is sufficient just to show a static picture of the mark. For other marks an annotated animation is needed before each one can be understood. Besides an animation, some marks need to show variations. Finally some marks, like menu marks, are best self-revealed incrementally. Depending on the characteristics of a mark, there are different ways of explaining the mark. This implies our self-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 marks and this information is intended to interactively teach them how to use these marks like an expert. No mark-based system that we know of supports this type of paradigm.

162

Chapter 7: Conclusions

7.1.   SUMMARY

This dissertation develops and evaluates an interaction technique called marking menus. Marking menus were developed based on several observations:

1) Marks can be an efficient and expressive way to issue commands, especially for pen-based computers.

2) Marks, by themselves, are not easy to use because unlike buttons, menus, and icons, they do not automatically reveal themselves to a user.

3) Therefore, marks must rely on some other interaction technique to reveal themselves to the user.

Given these observations we designed an interaction technique that combines menus and marks with the intention that using the menu helps a user learn the marks. The design of marking menus was based the design principles of self-revelation, guidance, and rehearsal. The principle of self-revelation states the system should interactively provide information about what commands are available and how to invoke those commands. The principle of guidance states that the way in which this information is provided should guide a user through invoking a command. The principle of rehearsal states that the guidance provided should be a rehearsal of act of drawing the mark associated with a command. The goal of these design principles is to help a user learn and use marks and quickly move from novice to expert.

163

After proposing a design for the technique based on these principles, we then evaluated the technique. The intention of the evaluation was to determine the limitations of the technique.

The first evaluation was an empirical experiment on non-hierarchic (i.e., one level) marking menus. This experiment showed that certain configurations of menu items make marking faster and less error-prone. Specifically, the experiment showed that four, eight, and twelve item menus enhance performance when marking. Also this experiment showed that subjects, on average, performed marks faster and more accurately with a mouse and stylus/tablet than with a trackball.

The second evaluation was a practical case study of two users' behaviors using a six-item marking menu for a real-life editing task. From this study we observed several things. First, with practice, users learn to use the marks and tend towards using the marks 100% of the time. Second, users utilized the features of the technique that were designed to aid in learning the marks (i.e., reselection and mark-confirmation). Third, using a mark in this situation was on average 3.5 times faster than selection using the menu.

A third evaluation was an empirical experiment examining the effect of menu breadth and depth on users' performance when selecting from hierarchic marking menus using marks. We found as breadth and depth of a menu structure increases, subject performance slows and the number of incorrect selections increases. Error rate appears to the limiting factor when selecting using marks. The experiment examined menus of breadth four, eight, and twelve, and menu depths from one to four. A significant change in error rate occurred when menu depth was greater than two and breadth was eight or twelve. The results suggest that marks can be used to reliably select from four-item menus up to four levels deep, or from eight-item menus up to two levels deep. This experiment also examined the effect of using a pen or a mouse. We found that subjects, on average, performed better with the pen than with the mouse. However, the difference in performance was not large. This indicated that the mouse would be an acceptable input device for hierarchic marking menus.

A final design study examined generalizing the design concepts of marking menus. Marking menus are an interaction technique that provides self-revelation, guidance, and rehearsal for a particular class of marks (i.e., straight lines and zig-zag marks).

164

We developed an interaction technique that provides self-revelation, guidance, and rehearsal for more general classes of marks. We also showed why the technique must differ from marking menus, and described an efficient means of implementing the technique.

7.2. CONTRIBUTIONS

The contributions of this work can be divided into two categories: contributions concerning marking menus specifically, and contributions concerning larger issues of human computer interaction.

7.2.1. Marking menus

The design of marking menus is a contribution in itself because of several design features. These features were described in detail in Section 2.2. The following is a summary of the design features that make marking menus a valuable and unique interaction technique. Marking menus:

- Allow menu selection acceleration without a keyboard.

- Permit acceleration on all menu items.

- Minimize the difference between the menu selection and accelerated selection.

- Permit pointing and menu selection acceleration with the same input device.

- Utilize marks that are easy and fast to draw.

- Use a spatial method for learning and remembering the association between menu items and marks.

- Are implementable as a "plug-in" software module.

The empirical studies and case studies in this work have contributed in:

Proving that users behave with marking menus as predicted. The design of marking menus features three modes of interaction: menu mode, mark confirmation mode, and mark mode. The case study in Chapter 4 has shown that users utilize all

three modes in the transition from novices, who use menus, to experts, who use marks. The case study also showed that users performed marks as quickly as keypresses. An equivalent interaction implemented with accelerator keys would have required pointing with the mouse *and* pressing an accelerator key. Hence we can conjecture that interaction was faster with marks than with accelerator keys in this setting.

Increasing our understanding of the limitations of marking menus. There is a limit to how accurately one can select items from a marking menu using a mark. The experiment in Chapter 5 has determined that selection using marks from menus with more than eight items per level and more than two levels of hierarchy will be error-prone. However, if two levels of eight item menus are used, marks can be used to quickly select from 64 menu items.

Determining configurations of marking menus that produce the best performance. Certain configurations of menu items make marking faster and less error-prone than other configurations. Specifically, our experiments have shown that 4, 6, 8 and 12 item menus and on-axis items enhance performance.

Demonstrating how command item selection and command parameters can be combined. Our case study demonstrates how both the starting point and end point of a mark can be used to express command parameters. This results in efficient interactions.

7.2.2. Issues of human computer interaction.

This work has several contributions to the study of human computer interaction in that it:

Identifies the fact that markings are not self-revealing. In the past, it has been assumed that mark-based interfaces will be easy to use because marks will be "natural" or mnemonic. This may be true in a some situations but not in all cases. There is a danger of falling into the trap that a system will be easy to use because it uses marks. This research makes the important point that while marks can be a very efficient means of interaction, this efficiency cannot be obtained if the user does not first have knowledge about the mark set. In some situations our experience with everyday pen and paper conventions supplies this knowledge. In other situations it

166

does not, and a self-revealing mechanism must be provided in conjunction with the marks.

Develops interaction techniques for self-revealing markings. Marking menus are a solution to the self-revealing problem for one particular class of mark. The crib-sheet/animator is a solution for more general classes of marks.

Identifies and develops the design principles of self-revelation, guidance and rehearsal. To solve the problem of marks not being self-revealing, this research develops the design principles of self-revelation, guidance, and rehearsal. Marking menus serves as an example of the application of the design principles and the crib-sheet/animator demonstrates that the principles can be applied to other situations. We feel that these design principles are valuable for interface design in general.

Develops a unique way to support novice/expert differences. The notions of guidance and rehearsal are a unique way of supporting novice/expert differences and transitions in mark-based interfaces. We know of no other systems that use a similar scheme.

Other research has dealt with novice/expert differences by providing explicit novice/expert modes. In these types of systems, novice mode has fewer functions than expert mode. The focus of this research is on supporting novice/expert differences and transitions using mark-based interfaces at the level of interaction, not at the level of available functions. These two approaches differ but they are not mutually exclusive.

Demystifies "the folk legend of gesture" in human computer interaction. It is clear from the literature that the types of gestures performed while operating an interface contribute to the overall sense of satisfaction with an interface. While others have observed that careful design of the body language of interactions results in better interface design, the research here is an explicit attempt to make use of this philosophy in a practical interaction technique.

Identifies the real value of marks as an interaction technique. Finally this research demonstrates that if the real advantages of particular interactions are understood, simple technology, used appropriately, can exploit these advantages. It is not simply the case that marks are desirable because marks are easy to remember. Another desirable property is the ability of a mark to efficiently express a command

and its parameters. The marks created by marking menus demonstrate this property. Furthermore, the technology required to support this property is not overly complex. Recognition methods, and ways of embedding and recognizing command parameters, are easily programmable.

7.3. FUTURE RESEARCH

As we developed marking menus we came across many interesting design variations, extensions and applications worth exploring:

• Adapt marking menus to be used on very small screens. A problem with very small screen computers is that there isn't enough room to draw long marks or display hierarchic menus. A variation on our marking menu design is to use a series of short strokes, all starting from the same location to perform a selection from a hierarchy of menus.

• Investigate other types of combinations of marks and menus. Continuous menu items, and dartboard and donut layouts, which were mentioned in Chapter 1, are examples of other types of combinations of marks and menus.

• Investigate feedback and pairing with command parameters. This research has only scratched the surface of things that can be done while performing a selection or after making a selection. Marking menus need the ability to show system status (e.g., display the current font), to preview the effects of selecting a menu item (e.g., highlighting a particular font in a menu causes an example of the font to be displayed), and to embed command parameters after a selection is confirmed (e.g., after selecting "volume" a user is automatically connected to a graphical slider). Integrating these features while maintaining the design principles is an open problem.

• 3D marking menus. Marking menus are based on selection by direction in two dimensions with two dimensional pointing devices. A natural generalization is to three dimensions.

• While our research has established some upper bounds on the limits of hierarchic marking menus, a natural extension would be a case study of user behavior with hierarchic marking menus in a real application. We know from our first case study on non-hierarchic menus that with enough practice users will use marks. Hierarchic

168

menus have many more menu items than non-hierarchic menus. For example, a menu hierarchy which is two levels deep, with eight items in each menu, contains 64 items. It would be interesting to see if this potential could be tapped in an application.

• Further development and evaluation of the crib-sheet/animator is another topic for future research. Clearly, user testing of the design is required. Also developing a parameterized version of the animator is an interesting research challenge.

• Investigating the application of self-revelation, guidance and rehearsal to other domains, besides marking is of interest. An example of the use of guidance and rehearsal in another domain is keyboard driven menus. The menus serve to reveal functionality to a novice, and the novice is guided through the menu by hitting keys to select menu items. This guidance provides a rehearsal of an expert type of behavior in which menu items are selected without looking or waiting for the menus to be displayed.

• There are many open questions concerning using marks and motor behavior. Does using a distinct gesture when drawing a mark have an advantage? What is a distinct gesture? Are there ways that we can design the gestures of drawing marks such that learning or performance is improved?

7.4. FINAL REMARKS

The interfaces to many ordinary, non-computerized objects have properties which make human operation of them second nature. For example, gear-shifts and turn-signal levers in automobiles have labels which we initially look at to learn the function mappings but with experience these mappings become automatic. Furthermore, with practice, the gestures of operating these devices become secondary to the task of driving. The fact that the gestures are unique contribute to our ability to perform them with very little attention. This provides the advantage of allowing our attention to be focused on other more important tasks, for example, watching traffic or reading street signs.

In this thesis, we have tried to exploit these types of properties in the realm of the computer interface. As computers become more entrenched as our everyday objects, tools and instruments, it is not unreasonable to expect them to exhibit the

169

properties that make many non-computerized objects easy and effective to use. This dissertation contributes to the understanding and creation of human computer interactions that have these properties.

170

References

Allen, R. B. (1983) Cognitive factors in the use of menus and trees: an experiment. *IEEE Journal on Selected Areas in Communications*, SAC 1(2), 333-336.

Apple Computer (1992) *Hypercard User's Guide.* Apple Computer, Cupertino, California.

Baecker, R., & Small, I. (1990) Animation at the interface. In Laurel, B. (Ed.)*The Art of Human-Computer Interface Design,* 251-267, Reading Massachusetts: Addison Wesley.

Baecker, R., Small, I., & Mander, R. (1991) Bringing icons to life. *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems,* 1-6, New York: ACM.

Baecker, R. M. (1969) Picture-driven animation. *Proceedings of the 1969 Spring Joint Computer Conference,* 273-278.

Barnard, B. J., & Grudin, J. (1988) Command Names. In Helander, M. (Ed.) *Handbook of Human Computer Interaction*, 237-255, B. V. North Holland: Elsevier Science.

Boritz, J., Booth, K. S., & Cowan, W. B. (1991) Fitts's law studies of directional mouse movement. *Proceedings of Graphics Interface '91,* 216-223.

Bush, V. (1945) As We May Think. *Atlantic Monthly.* July, 101-108.

Buxton, W. (1986). Chunking and phrasing and the design of human-computer dialogues. In Kugler, H. J. (Ed.) *Information Processing '86, Proceedings of the IFIP 10th World Computer Congress*, 475-480, Amsterdam: North Holland Publishers.

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

Callahan, J., Hopkins, D., Weiser, M., & Shneiderman, B. (1988) An empirical comparison of pie vs. linear menus. *Proceedings of the CHI '88 Conference on Human Factors in Computing Systems,* 95-100, New York: ACM.

171

Card S. K. (1982) User perceptual mechanisms in the search of computer command menus. *Proceedings of the CHI '82 Conference on Human Factors in Computing Systems,* 190-196, New York: ACM.

Card, S. K., Moran, T. P., & Newell, A. (1983) *The Psychology of Human-Computer Interaction.* Hillsdale NJ: Lawerence Erlbaum.

Card, S. K., Robertson, G. G., & Mackinlay, J. D., (1991) The information visualizer, an information workspace. *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems,* 181-188, New York: ACM.

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

Carroll, J. M., & Carrithers, C. (1984) Training wheels in a user interface. *Communications of the ACM,* 27, 800-806.

Coleman, M. L. (1969) Text Editing on a Graphics Display Device Using Hand-drawn Proofreader's Symbols. *Proceedings of the Second University of Illinois Conference on Computer Graphics.* 282-291, Chicago: University of Illinois Press.

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

Ellis, T. O., & Sibley, W. L. (1967) On the Development of Equitable Graphic I/O. *IEEE Transactions on the Human Factors in Electronics.* 8(1), 15-17.

Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., Lee, D., McCall, K., Pedersen, E., Pier, K., Tang, J., & Welch, B. (1992) Liveboard: A large interactive display supporting group meetings. presentations and remote collaboration. *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems,* 599-607, New York: ACM.

Fischman, M. G. (1984) Programming time as a function of number of movement parts and changes in movement direction. *Journal of Motor Behavior,* 16(4), 405-423.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology,* 47, 381-391.

Furnas, G., Gomez, L., Landauer, T., & Dumais, S. (1982) Statistical Semantics: How can a computer use what people name things to guess what things people mean when they name things? *Proceedings of the CHI '82 Conference on Human Factors in Computing Systems,* 251-253, New York: ACM.

Gaver, W., W. (1991) Technology affordances, *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems,* 79-84, New York: ACM.

172

Gibson, J. J. (1979) *The ecological approach to visual perception.* Houghton Mifflin, New York.

Gibson, J. J. (1982) *Reasons for realism: Selected essays of James J. Gibson.* Reed, E. & Jones, R. (Ed.), Hillsdale NJ: Lawerence Erlbaum.

Go (1991) *PenPoint System Manual,* Go Corporation, Foster City, CA.

Goldberg D., & Goodisman A. (1991) Stylus User Interfaces for Manipulating Text. *Proceedings of the ACM Symposium on User Interface Software and Technology,* 127-135, New York: ACM.

Gould, J. D., & Salaun, J. (1987) Behavioral Experiments in Handmarks. *Proceedings of the CHI + GI '91 Conference on Human Factors in Computing Systems and Graphics Interface,* 175-181, New York: ACM.

Guiard, Y., Diaz, G., & Beaubaton, D. (1983) Left-hand advantage in right-handers for spatial constant error: preliminary evidence in a unimanual ballistic aimed movement. *Neuropsychologia,* Vol. 21, No. 1, 111-115.

Hardock, G. (1991). Design issues for line driven text editing/annotation systems. *Proceedings of the Graphics Interface '91 Conference,* 77-84, Toronto: Canadian Information Processing Society.

Hoeber, T. (1988) Face to face with Open Look, Byte, V(13) (Dec. '88), 286-288.

Hopkins, D. (1987) Direction selection is easy as pie menus! *login: The USENIX Association Newsletter,* 12(5), 31-32.

Hopkins, D. (1991) The design and implementation of pie menus. *Dr. Dobb's Journal,* 16(12), 16-26.

Hornbuckle, G. D. (1967) The Computer Graphics User/Machine Interface. *IEEE Transactions on the Human Factors in Electronics.* 8(1), 17-20.

Jorgensen, A. H., Barnard, P., Hammond, N., and Clark, I., (1983) Naming commands: An analysis of designers' naming behavior. *Psychology of computer use,* Green T. R. G., Payne S. J., and van derr Veer, G. C. (Eds.), 69-88, London: Academic Press.

Keele, S. W. (1968) Movement control in skilled motor performance, *Psychological Bulletin,* 70, 387-403.

Kiger, J. L. (1984) The depth/breadth tradeoff in the design of menu-driven user interfaces. *International Journal of Man Machine Studies,* 20, 210-213.

Kirk, R. E. (1982) *Experimental design: Procedures for the Behavioral Sciences.* Belmont California: Wadsworth.

Krueger M. W., Giofriddo T., & Hinrichsen K. (1985) VIDEOPLACE — An Artificial Reality. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems,* 35-40, New York: ACM.

Kurtenbach, G. & Baudel, T. (1992) HyperMark: Issuing commands by drawing marks in Hypercard. *Proceedings of CHI '92 Conference poster and short talks,* 64, New York: ACM.

Kurtenbach, G. & Buxton W. (1991) Issues in combining marking and direct manipulation techniques. *Proceedings of UIST '91 Conference,* 137-144, New York: ACM.

Kurtenbach, G. & Buxton, W. (1991) GEdit: A testbed for editing by contiguous gesture. *SIGCHI Bulletin,* 22-26, New York: ACM.

Kurtenbach, G. & Buxton, W. (1993) The limits of expert performance using hierarchical marking menus. to appear in *Proceedings of the CHI '93 Conference on Human Factors in Computing Systems,* New York: ACM.

Kurtenbach, G. & Hulteen, E. (1990) Gesture in Human-Computer Communication. In Laurel, B. (Ed.)*The Art of Human-Computer Interface Design,,* 309-317, Reading Massachusetts: Addison Wesley.

Kurtenbach, G., Sellen, A., & Buxton, W. (1993) An empirical evaluation of some articulatory and cognitive aspects of "marking menus". *Human Computer Interaction ,* 8(2), 1-23

Landauer, T. K. & Nachbar, D. W. (1985) Selection from alphabetic and numeric trees using a touch screen: breadth, depth and width. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems,* 73-78, New York: ACM.

Lee, E. & MacGregor, J. (1985) Minimizing user search time in menu driven systems. *Human Factors,* 27(2), 157-162.

Licklider, J. C. R. (1960) Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics.* March 1960, 4-11.

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

Mackenzie, I. S. & Buxton, W. (1992) Extending Fitts' law to two-dimensional tasks. *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems,* 219-226, New York: ACM.

Mackenzie, I. S., Sellen, A. J., and Buxton, W. (1991) A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of the CHI '91 Conference on Human Factors in Computing Systems,* 161-166, New York: ACM.

174

Makuni, R. (1986) Representing the Process of Composing Chinese Temples. *Design Computing.* Vol. 1, 216-235.

Malfara, A. & Jones, B. (1981) Hemispheric asymmetries in motor control of guided reaching with and without optic displacement. *Neuropsychologia,* Vol. 19, No. 3, 483-486.

McDonald, J. E., Stone, J. D., & Liebelt, L. S. (1983) Searching for items in menus: The effects of organization and type of target. *Proceedings of Human Factors Society 27th Annual Meeting.* 834-837, Santa Monica, CA: Human Factor Society.

Momenta, (1991) *Momenta User's Reference Manual.* Momenta, 295 North Bernardo Avenue, Mountain View, California.

Morrel-Samuels, P. (1990) Clarifying the distinction between lexical and gestural commands. *International Journal of Man-Machine Studies ,* 32, 581-590.

Nilsen, E. L. (1991) *Perceptual-motor control in human-computer interaction.* Technical Report No. 37, University of Michigan, Cognitive Science and Machine Intelligence Laboratory.

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

Norman, D. A. (1981) Categorization of action slips. *Psychological Review,* 88, 1-15.

Normile, D. & Johnson, J. T. (1990). Computers without keys. *Popular Science,* August 1990, 66-69.

Paap, K. R. & Roske-Hofstrand, R. J. (1986) The optimal number of menu options per panel. *Human Factors,* 28(4), 1-12.

Paap, K. R. & Roske-Hofstrand, R. J. (1988) Design of Menus. *Handbook of Human Computer Interaction,* Helander, M. (Ed.), 205-235, B. V. North Holland: Elsevier Science.

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

Perkins R., Blatt L. A., Workman D., & Ehrlich S. F. (1989) Interactive tutorial design in the product development cycle. *Proceeding of the Human Factors Society 33rd Annual Meeting,* 268-272.

Perlman, G. (1984) Making the right choices with menus. *Proceedings of Interact '84,* 317-320, B. V. North Holland: Elsevier Science.

Rasmussen, J. (1983) Skills, Rules and Knowledge: Signals, Signs and Symbols and other Distinctions in Human Performance Models. *IEEE Transactions on Systems, Man and Cybernetics,* SMC-13, 257-266.

Rebello, K. (1990) New PCs can kiss keyboards good-bye. *USA Today*, Feb. 22., 6B.

Rhyne, J. R. & Wolf, C. G. (1986) *Gestural Interfaces for Information Processing Applications.* IBM Technical Report 12179 (#54544).

Rhyne, J. R. (1987) Dialogue Management for Gestural Interfaces. *ACM Computer Graphics.* 21(2), 137-142.

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

Rubine, D. (1991) Specifying Gestures by Example. *Computer Graphics,* 25(4), 329-337.

Rubine, D. H. (1990) *The Automatic Recognition of Gestures.* Ph.D. Thesis, Dept. of Computer Science, Carnegie Mellon University.

Sellen, A. J., Kurtenbach, G., & Buxton, W. (1992) The prevention of mode errors through sensory feedback. *Human-Computer Interaction.* 7(2), 141-164.

Sellen, A. J. & Nicol, A. (1990) Building user-centered on-line help. In Laurel, B. (Ed.)*The Art of Human-Computer Interface Design,* 143-153, Reading Massachusetts: Addison Wesley.

Sellen, A. J. (1992) Speech patterns in video-mediated conversation, *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems,* 49-59, New York: ACM.

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

Sibert, J., Buffa, M. G., Crane, H. D., Doster, W., Rhyne, J. R., & Ward, J. R. (1987) Issues Limiting the Acceptance of User Interfaces Using Gestures Input and Handwriting Character Recognition. *Proceedings of the CHI + GI '91 Conference on Human Factors in Computing Systems and Graphics Interface,* 155-158, New York: ACM.

Snoddy, G. S. (1926) Learning and stability. *Journal of Applied Psychology* 10, 1-36.

Snowberry, K., Parkinson, S. R., & Sisson, N. (1983) Computer display menus. *Ergonomics,* 26(7), 699-712.

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

Sukaviriya, P. (1988) Dynamic construction of animated help from application context. *Proceedings of the ACM Symposium on User Interface Software and Technology '88,* 190-202, New York: ACM.

Sutherland, I. E. (1963) Sketchpad: A man-machine graphical communication system. *AFIPS Conference Proceedings 23,* 329-346.

Walker, N., Smelcer, J. B., & Nilsen, E. (1991) Optimizing speed and accuracy of menu selection: a comparison of walking and pull-down menus. *International Journal of Man-Machine Studies,* 35, 871-890.

Ward, J. R. & Blesser, B. (1985) Interactive Recognition of Handprinted Characters for Computer Input. *IEEE Computer Graphics & Algorithms .* Sept. 1985, 24-37.

Weiser, M. (1991) The computer for the 21st century. *Scientific American,* 265(3), 94-104.

Welbourn, L. K. & Whitrow, R. J. (1988) A gesture based text editor. *People and Computers IV, Proceedings of the Fourth Conference of the British Computer Society Human-Computer Specialist Group.* 363-371, Cambridge UK: Cambridge University Press.

Westheimer, G. & McKee, S. P. (1977) Spatial configurations for visual hyperacuity. *Vision Research,* 17, 941-947.

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

Wixon, D., Whiteside, J., Good, M., & Jones, S. (1983) Building a user-defined interface. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems,* 185-191, New York: ACM.

Wolf, C. G. & Morrel-Samuels, P. (1987) The use of hand-drawn gestures for text editing. *International Journal of Man-Machine Studies ,* 27, 91-102.

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

Wolf, C. G., Rhyne, J. R., & Ellozy, H. A., (1989). The paper-like interface. *Designing on Using Human Computer Interface and Knowledge-Based Systems.* 494-501, B. V. North Holland: Elsevier Science.

X11 (1988) *X window system user's guide for version 11.* X window series, v. 3, Sebastopol CA: O'Reilly & Associates._

177

Appendix A: Statistical Methods

This appendix explains the statistical methods used in this dissertation. Analysis of variance (ANOVA) is used for hypothesis testing. Specifically, the F-statistic is used to determine if an *independent variable* has any effect on a *dependent variable*. In an experiment, the dependent variable is a variable being measured. The independent variable is a variable being controlled.

Testing for differences in means: $F(k - 1, k(n - 1)) = f$, $p < \alpha$.

Data is grouped according to different values of the independent variable. Each group is commonly referred to as a *treatment*. Random samples of size n are selected from each of k treatments. It is assumed that the k treatments each have a population that is independent and normally distributed with means $\mu_1, \mu_2, \ldots, \mu_k$ and a common variance σ^2. The null hypothesis can be represented as:

$$\mu_1 = \mu_2 = \ldots = \mu_k$$

The ANOVA procedure separates the total variability of the samples into two component: s_1^2 and $s^{2.}$ The variance s_1^2 is the variability between treatments attributed to changes in the independent variable and chance or random variation. The variance s^2 is the variability within treatments due to chance or random variation.

It can be shown that, assuming the null hypothesis is true, the ratio:

$$f = s_1^2/s^{2.}$$

is a value of the random variable F having the F distribution with $k - 1$ and $k(n - 1)$ degrees of freedom. Since s_1^2 overestimates the true variance when the null hypothesis is false, a large value for f suggest a large portion of the variance in the

179

dependent variable is caused by the independent variable. A test can be done by comparing the observed value f with the theoretical value of $F(k - 1, k(n - 1))$ and reporting the probability, p, of such a large value for f occurring simply by chance. If p is very small (e.g., $p < .05$), this suggests that the null hypothesis should be rejected.

Multiple comparison of means: Tukey HSD, $\alpha = p$

After determining a significant f ratio, it is may be necessary to determine which pairs of means are significantly different. Various procedures, which are referred to as post-hoc comparisons, allow this. If means μ_1 and μ_2 are being compared, the null hypothesis is:

$$\mu_1 - \mu_2 = 0.$$

A Tukey HSD post-hoc test reports the significantly differing means with a probability of α of incorrectly rejecting the null hypothesis (i.e., no difference exists between the means). Generally a .05 level of significance is used. This means one can be 95% sure that two means actually differ.

Contrasting means: $F(1) = f, p < \alpha$.

Post-hoc tests are not available for within subjects factors in repeated measures experimental design. An alternative method for determining which pairs of means are significantly different is by contrasting means. ANOVA separates the variance into two components: SSw and s^2. SSw is the variance attributed to the difference between the means. The variance s^2 is the variability due to chance or random variation.

It can be shown that, assuming the null hypothesis is true, the ratio:

$$f = SSw/s^2.$$

is a value of the random variable F having the F distribution with 1 and $n - k$ degrees of freedom. Since SSw overestimates the true variance when the null hypothesis is false, large values of f indicate a large portion of the variance is due to a difference between the means. A test can be done by comparing the observed value f with the theoretical value of $F(1, n - k))$ and reporting the probability, p, of such a large value

180

for f occurring simply by chance. If p is very small (e.g., $p < .05$), this suggests that the null hypothesis should be rejected.

Testing for linear relationships: F(1, n - 2)

The *F*-statistic is used to provide a single significance probability of a linear relationship between dependent and independent variables. In this case, the null hypothesis is that the slope of the regression line is zero. If the null hypothesis is true, then

$$f = SSR/s^2.$$

Where *SSR* is the amount of variation explained by the straight regression line. The variance s^2 is the variability around the regression line due to errors. It can be shown f is the value of the random variable F having the F distribution with 1 and n - 2 degrees of freedom. A test can be done by comparing the observed value f with the theoretical value of $F(1, n - 2))$ and reporting the probability, p, of such a large value for f occurring simply by chance. If p is very small (e.g., $p < .05$), this suggests that the null hypothesis should be rejected.

Testing a linear relationship for goodness of fit: r^2

The sample correlation coefficient r^2 is used to test the quality of the fit of a linear regression line. The amount of variation in the dependent variable which is explained by the independent variable is $r^2 \times 100\%$. A r^2 value greater than .5 is considered to indicate a linear relationship.

For further information on these statistical methods see Kirk (1982).

181

Don Hopkins
Don Hopkins' Web Site

[Search field] Search

Syndicate

User login

Username: *

[input field]

Password: *

[input field]

[Log in]

- Request new password

ITS.Svensson.org

Home » topics » Software » Pie Menus » Pie Menu Applications

The Design and Implementation of Pie Menus -- Dr. Dobb's Journal, Dec. 1991

Submitted by dhopkins on Tue, 2005-09-27 00:34.   NeWS | Pie Menu Applications | Pie Menu Design

The Design and Implementation of Pie Menus

There're Fast, Easy, and Self-Revealing.

Copyright (C) 1991 by Don Hopkins.
Originally published in Dr. Dobb's Journal, Dec. 1991, lead cover story, user interface issue.

Introduction

Although the computer screen is two-dimensional, today most users of windowing environments control their systems with a one-dimensional list of choices -- the standard pull-down or drop-down menus such as those found on Microsoft Windows, Presentation Manager, or the Macintosh.

This article describes an alternative user-interface technique I call "pie" menus, which is two-dimensional, circular, and in many ways easier to use and faster than conventional linear menus. Pie menus also work well with alternative pointing devices such as those found in stylus or pen-based systems. I developed pie menus at the University of Maryland in 1986 and have been studying and improving them over the last five years.

During that time, pie menus have been implemented by myself and my colleagues on four different platforms: X10 with the uwm window manager, SunView, NeWS with the Lite Toolkit, and OpenWindows with the NeWS Toolkit. Fellow researchers have conducted both comparison tests between pie menus and linear menus, and also tests with different kinds of pointing devices, including mice, pens, and trackballs.

Included with this article are relevant code excerpts from the most recent NeWS implementation, written in Sun's object-oriented PostScript dialect.

Pie Menu Properties

In their two-dimensional form, pie menus are round menus containing menu items positioned around the cursor -- as opposed to the rows or columns of traditional linear menus. The menu item target regions are shaped like the slices of a pie, and the cursor starts out in the center, in a small inactive region. The active regions are all adjacent to the cursor, but each in a different direction. You select from a pie menu by clicking the mouse or tapping the stylus, and then pointing in a particular direction.
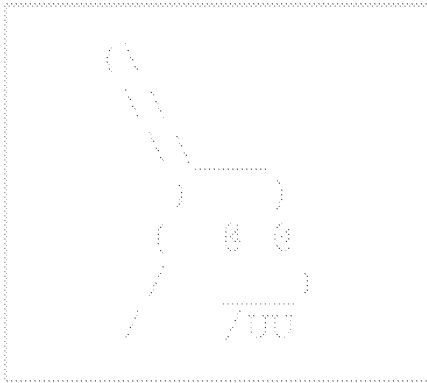
Although there are multiple kinds of pie menus, the most common implementation uses the relative direction of the pointing device to determine the selection -- as compared with the absolute positioning required by linear menus. The wedge-shaped slices of the pie, adjacent to the cursor but in different direction, correspond to the menu selections. Visually, feedback is provided to the user in the form of highlighting the wedge-shaped slices of the pie. In the center of the pie, where the cursor starts out, is an inactive region.

When a pie menu pops up, it is centered at the location of the click that invoked it: where the mouse button was pressed (or the screen was touched, or the pen was tapped). The center of the pie is inactive, so clicking again without moving dismisses the menu and selects nothing. The circular layout minimizxes the motion required to make a selection. As the cursor moves into the wider area of a slice, you gain leverage, and your control of direction improves. To exploit this property, the active target areas can extend out to the edges of the screen, so you can move the cursor as far as required to select precisely the intended item.

You can move into a slice to select it, or move around the menu, reselecting another slice. As you browse around before choosing, the slice in the direction of the cursor is highlighted, to show what will happen if you click (or, if you have the button down, what will happen if you release it). When the cursor is in the center, none of the items are highlighted, because that region is inactive.

Pie menus can work with a variety of pointing devices -- not just mice, but also pens, trackballs, touchscreens, and (if you'll pardon the hand waving) data gloves. The look and feel should, of course, be adapted to fit the qualities and constraints of the particular device. For example, in the case of the data glove, the two-dimensional circle of a pie could become a three-dimensional sphere, and the wedges could become cones in space.

In all cases, a goal of pie menus is to provide a smooth, reliable gestural style of interaction for novices and experts.

Pie Menu Advantages

Pie menus are faster and more reliable than linear menus, because pointing at a slice requires very little cursor motion, and the large area and wedge shape make them easy targets.

For the novice, pie menus are easy because they are a self-revealing gestural interface: They show what you can do and direct you how to do it. By clicking and popping up a pie menu, looking at the labels, moving the cursor in the desired direction, then clicking to make a selection, you learn the menu and practice the gesture to "mark ahead" ("mouse ahead" in the case of a mouse, "wave ahead" in the case of a dataglove). With a little practice, it becomes quite easy to mark ahead even through nested pie menus.

For the expert, they're efficient because -- without even looking -- you can move in any direction, and mark ahead so fast that the menu doesn't even pop up. Only when used more slowly like a

traditional menu, does a pie menu pop up on the screen, to reveal the available selections.

Most importantly, novices soon become experts, because every time you select from a pie menu, you practice the motion to mark ahead, so you naturally learn to do it by feel! As Jaron Lanier of VPL Research has remarked, "The mind may forget, but the body remembers." Pie menus take advantage of the body's ability to remember muscle motion and direction, even when the mind has forgotten the corresponding symbolic labels.

By moving further from the pie menu center, a more accurate selection is assured. This feature facilitates mark ahead. Our experience has been that the expert pie menu user can easily mark ahead on an eight-item menu. Linear menus don't have this property, so it is difficult to mark ahead more than two items.

This property is especially important in mobile computing applications and other situations where the input data stream is noisy because of factors such as hand jitter, pen skipping, mouse slipping, or vehicular motion (not to mention tectonic activity).

There are particular applications, such as entering compass directions, time, angular degrees, and spatially related commands, which work particularly well with pie menus. However, as we'll see further on, pies win over linear menus even for ordinary tasks.

Pie Menu Flavors

There are many flavors or variants of pie menus. One obvious variation is to use the semicircular pie ("fan") menus at the edge of the screen.

Secondly, although the usual form of pie menus is to use only the directional angle in determining a selection, there is a vartiant of pie menus which offers two parameters of choice with a single user action. In this case, both the direction and the distance between the two points are used as parameters to the selection. The ability to specify two input parameters at once can be used in situations where the input space is two-dimensional. Direction and distance may be discrete or continuous, as appropriate.

For example, for a graphics or word processing application, a dual-parameter pie menu allows you to specify both the size and style of a typographic font in one gesture. The direction selects the font style from a set of possible attributes, and the distance selects the point size from the range of sizes. An increased distance from the center corresponds to an increase in the point size. This pie menus provides satisfying visual feedback by dynamcally shrinking and swelling a text sample in the menu center, as the user moves the pointer in and out.

Other variants include scrolling spiral pies, rings, pies within square windows, and continuous circular fields. These variants are discussed in a later section.

A minor variation in the use of pie menus is whether you click-and-

drag as the menu pops up, or whether two clicks are required: one to make the menu appear, another to make the selection. In fact, it's possible to support both.

Pie Menu Implentations

As mentioned earlier, several pie menu implementations exist, including: X10, SunView, and two NeWS implementation (using different toolkits).

I first attempted to implement pie menus in June 1986 on a Sun 3/160 running the X10 window system by adding them to the "uwm" window manager. The user could define nested menus in a ".uwmrc" file and bind them to mouse buttons. The default menu layout was specified by an initial angle and a radius that you could override in any menu whose labels overlapped. The pop-up menu was rectangular, large enough to hold the labels, and had a title at the top.

Then I linked the window manager into Mitch Bradley's Sun Forth, to make a Forth-extensible window manager with pie menus. I used this interactively programmable system to experiment with pie menu tracking and window management techniques, and to administer and collect data for Jack Callahan's experiment comparing pie menus with linear menus.

In January 1987, while snowed in at home, Mark Weiser implemented pie menus for the SunView window system. They are featured in his reknowned "SDI" game, the source code for which is available free of charge.

I implemented pie menus in round windwos for the Lite Toolkit in NeWS 1.0 in May 1987. The Lite Toolkit is implemented in NeWS, Sun's object-oriented PostScript dialect. Pie menus are built on top of the abstract menu class, so they have the same application program interface as linear menus. Therefore, pie menus can transparently replace the default menu class, turning every menu in the system into a pie, without having to modify other parts of the system or applications.

Because of the equivalence in semantics between pie menus and linear menus, pies can replace linear menus in systems in which menu processing can be revectored. Both the Macintosh and Microsoft Windows come to mind as possible candidates for pie menu implementations. Of course, for best results, the application's menus should be arranged with a circular layout in mind.

My most recent implementation of pie menus runs under the NeWS Toolkit, the most modern object-oriented toolkit for NeWS, shipped with Sun Open Windows, Version 3. The pie menu source code, and several special-purpose classes, as well as sample applications using pie menus are all available for no charge.

Usability Testing

Over the years, there have been a number or research projects studying the human factors aspects of pie menus.

Jack Callahan's study compares the seek time and error rates in pies versus linear menus. There is a hypothesis known as Fitts' law, which states that the "seek time" required to point the cursor at the target depends on the target's area and distance. The wedge-shaped slices of a pie menu are all large and close to the cursor, so Fitts' law predicts good times for pie menus. In comparison, the rectangular target areas of a traditional linear menu are small, and each is placed at a different distance from the starting location.

Callahan's controlled experiment supports the result predicted by Fitt's law. Three types of eight-item menu task groupings were used: Pie tasks (North, NE, East, and so on), linear tasks (First, Second, Third, and so on), and unclassified tasks (Center, Bold, Italic, and so on). Subjects with little or no mouse experience were presented menus in both linear and pie formats, and told to make a certain selection from each. Those subjects uising pie menus were able to make selection significantly faster and with fewer errors for all three task groupings.

The fewer the items, the faster and more reliable pie menus are, because of their bigger slices. But other factors contribute to their efficiency. Pies with an even number of items are symmetric, so the directional angles are convenient to remember and articulate. Certain numbers of items work well with various metaphors, such as a clock, an on/off switch, or a compass. Eight-item pies are optimal for many tasks: They're symmetric, evenly divisible along vertical, horizontal, and diagonal axes, and have distinct, well-known directions.

Gordon Kurtenbach carried out an experiment comparing pie menus with different visual feedback styles, numbers of slices, and input devices. One interesting result was that menus with an even number of items were generally better than those with odd numbers. Also, menus with eight items were especially fast and easy to learn, because of their primary and secondary compass directions. Another result of Kurtenbach's experiment was that, with regard to speed and accuracy, pens were better than mice, and mice were better than trackballs.

The "Eight Days a Week" menu shown in Figure 1 is a contrived example of eight-item symmetry: It has seven items for the days of the week, plus one for today. Monday is on the left, going around counterclockwise to Friday on the right. Wednesday is at the bottom, in the middle of the week, and the weekend floats above on the diagonals. Today is at the top, so it's always an easy choice. The NeWS Toolkit code that creats this pie menu is shown in Listing 1.
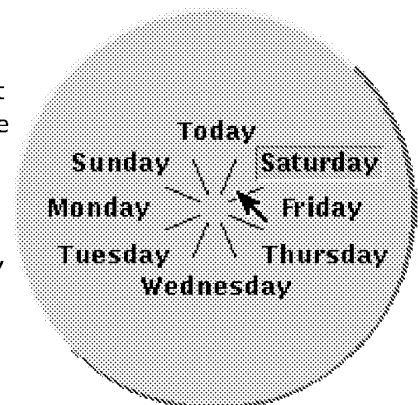
Pie Menu Disadvantages

The main disadvantage of pie menus is that when they pop up, they can take a lot of screen space due to their circular layout. Long item labels can make then very large, while short labels or small icons make them more compact and take up less screen space.

The layout algorithm should have three goals: to minimize the menu size, to prevent menu labels from overlapping, and to clearly associate labels with their direction. It's not necessary to confine each label to the interior of its slice -- that could result in enormous menus. In a naive implementation, you might use text labels rotates around the center of the pie. But rotated text turns out not to work well, because it exaggerates "jaggies". This is hard to read without rotating your head, and doesn't even satisfy the goal of minimizing menu size.

One successful layout policy I've implemented justifies each label edge within its slice, at an inner radius big enough that no two adjacent labels overlap. To delimit the target areas, short lines are drawn between the slices, inside the circle of labels, like cuts in a pie crust.

One solution to the problem of pie menus with too many items is to divide up large menus into smaller, logically related submenus. Nested pies work quite well, as you can mark ahead quickly through several levels. You remember the route through the menus in the same way you remember how to drive to a friend's house: by going down familiar roads and making the correct turn at each intersection.

Another alternative is to use a scrolling pie menu that encompasses many items in a spiral but only displays a fixed number of them at once. By winding the cursor around the menu center, you can scroll through all the items, like walking up or down a spiral staircase.

Other Design Considerations

When you mark ahead quickly to select from a familiar pie, it can be annoying if the menu pops up after you've already finished the selection, and then pops down, causing the screen to repaint and slowing down interaction. If you don't need to see the menu, it shouldn't show itself. When you mark ahead, interaction is much quicker if the menu display is preempted while the cursor is in motion, so you never have to stop and wait for the computer to catch up. If you click up a menu when the cursor is at rest, it should pop up immediatly, but if you press and move, the menu should not display until you sit still. If you mark ahead, selecting with a smooth continuous motion, the menu should not display at all. However, it's quite helpful to give some type of feedback, such as displaying the selected label on an overlay near the cursor, or previewing the effect of the selection.

When you pop up a pie menu near the edge of the screen, the menu may have to be moved by a certain offset in order to fit completely on the screen, otherwise you couldn't see or select all the items. But it would be quite unexpected were the menu to slip out from under the click, leaving the cursor pointing at the wrong
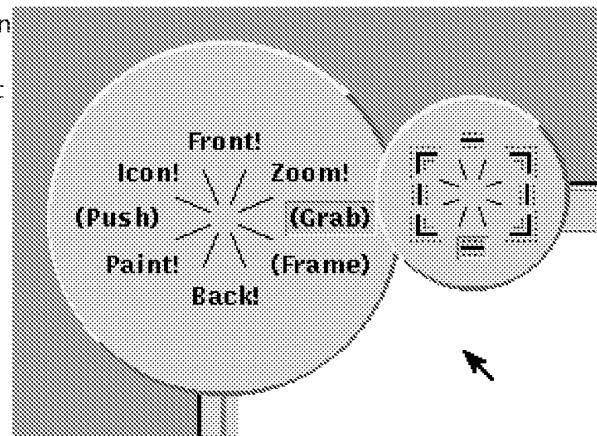
slice. So whenever the menu is displayed on the screen, and it must be moved in order to fit, it is important to "warp" the cursor by the same offset, relative to its position at the time the menu is displayed. If you mark ahead so quickly that the menu display is preempted, the cursor shouldn't be warped. Pen- and touchscreen-based pie menus can't warp your pen or finger, so pie menus along the screen edge could pop up as semicircular fans. Note that cursor warping is also an issue that linear menus should address.

Ideally, pie menu designers should arrange the labels and submenus in directions that reflect spatial associations and relationships between them, making it easy to remember the directions. Complementary items can be opposite each other, and orthogonal pairs at right angles.

It's difficult to mark ahead into a pie menu whose items are not always in the same direction, because if the number of items changes, and they move around, you never know in which directions to expect them. Pie menus are better for selecting from a constant set of items, such as a list of commands, and best when the items are thoughtfully arranged to exploit the circular layout.

Sample Pie Menu

The pie menu shown in Figure 2 is an example of one that I added to the NeWS environment. Clicking on the window frame pops up this menu of window-management commands, designed to take advantage of mark ahead. Because this menu is so commonly used, you can learn to use it quickly, and save a lot of time. At the left of the figure is the top-level menu with commonly used commands and logically related submenus. The "Grab" item has been selected, popping up a graphical submenu of corners and edges. The icon for the bottom edge is highlighted, but has noot yet been selected. Clicking in that slice allows you to grab and stretch the edge of the window frame.

Figure 3 shows a second example, a color wheel that allows you to set the brightness, and to select a color from a continuous range of hues and saturations. The hue varies smoothly around the color wheel with direction, and the saturation varies smoothly with distance, with pure colors in the center fading to gray around the edge. Outside the pale perimeter is a continuous band of grays from white to black, that looks like the shadow inside a paint can, and functions as a circular brightness dial. Dipping into this gray border sets the brightness of the whole wheel. You may select any shade

of gray around the border, or move back into the paint can, to select a color at the current brightness. As you move around, the cursor shows the true color selected, and because the cursor is displayued even before the menu is popped up, you can mark ahead and select a color without popping up the menu!
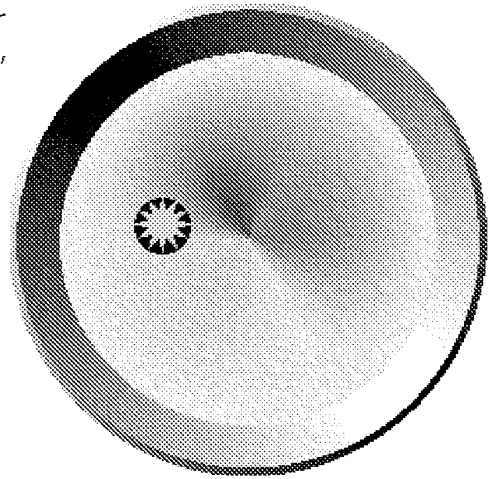
Conclusion

Pie menus are easy to learn, fast to use, and provide a gestural style of interaction that suits both novices and experts. The techniques are available for anyone to share, so take a look and feel free!

» **dhopkins's blog** | **Login** to post comments

Bloomberg Businessweek

Innovation & Design

http://www.businessweek.com/stories/2008-01-02/the-long-nose-of-innovationbusinessweek-business-news-stock-market-and-financial-advice

The Long Nose of Innovation

By Bill Buxton January 02, 2008

The bulk of innovation is low-amplitude and takes place over a long period. Companies should focus on refining existing technologies as much as on creation

In October of 2004, Chris Anderson wrote an article in Wired magazine called The Long Tail, a theory he expanded upon in his 2006 book, The Long Tail: Why the Future of Business is Selling Less of More. In it he captures some interesting attributes of online services, using a concept from statistics which describes how it is now possible for the "long tail" of a low-amplitude population to make up the majority of a company's business.

One of his examples came from music: A large quantity of often obscure but nonetheless listened-to music can outperform a much smaller quantity of huge hits. The implications of the phenomenon have been significant for those interested in understanding the meaningful attributes of online vs. brick-and-mortar businesses and the book has apparently had an enormous impact among executives and entrepreneurs.

But those looking to apply the theory to the implementation of innovation within an organization should beware. My belief is there is a mirror-image of the long tail that is equally important to those wanting to understand the process of innovation. It states that the bulk of innovation behind the latest "wow" moment (multi-touch on the iPhone, for example) is also low-amplitude and takes place over a long period—but well before the "new" idea has become generally known, much less reached the tipping point. It is what I call The Long Nose of Innovation.

A Mouse Family Tree

As with the Long Tail, the low-frequency component of the Long Nose may well outweigh the later high-frequency and (more likely) high-visibility section in terms of dollars, time, energy, and imagination. Think of the mouse. First built in around 1965 by William English and Doug Engelbart, by 1968 it was copied (with the originators' cooperation) for use in a music and animation system at the National Research Council of Canada. Around 1973, Xerox PARC adopted a version as the graphical input device for the Alto computer.

In 1980, 3 Rivers Systems of Pittsburgh released their PERQ-1 workstation, which I believe to be the first commercially available computer that used a mouse. A year later came the Xerox Star 8010 workstation, and in January, 1984, the first Macintosh—the latter being the computer that brought the mouse to the attention of the general public. However it was not until 1995, with the release of Windows 95, that the mouse became ubiquitous.

On the surface it might appear that the benefits of the mouse were obvious—and therefore it's surprising it took 30 years to go from first demonstration to mainstream. But this 30-year gestation period turns out to be more

typical than surprising. In 2003 my office mate at Microsoft (MSFT), Butler Lampson, presented a report to the Computer Science and Telecommunications Board of the National Research Council in Washington which traced the history of a number of key technologies driving the telecommunications and information technology sectors.

Understanding Immature Technologies

The report analyzed each technology (time-sharing, client/server computing, LANs, relational databases, VLSI design, etc.) from first inception to the point where it turned into a billion dollar industry. What was consistent among virtually all the results was how long each took to move from inception to ubiquity. Twenty years of jumping around from university labs to corporate labs to products was typical. And 30 years, as with the mouse and RISC processors, was not at all unusual (and remember, this is the "fast-paced world of computers," where it is "almost impossible" to keep up).

Any technology that is going to have significant impact over the next 10 years is already at least 10 years old. That doesn't imply that the 10-year-old technologies we might draw from are mature or that we understand their implications; rather, just the basic concept is known, or knowable to those who care to look.

Here's the message to be heeded: Innovation is not about alchemy. In fact, innovation is not about invention. An idea may well start with an invention, but the bulk of the work and creativity is in that idea's augmentation and refinement. The newer the idea, the coarser the granularity of most analysis, and the more likely people are to say, "oh, that's just like X" or "that's been done before," without any appreciation for how much work and innovation is involved in taking an idea from concept to wide practice.

Rewarding the Art of Refinement

The heart of the innovation process has to do with prospecting, mining, refining, and goldsmithing. Knowing how and where to look and recognizing gold when you find it is just the start. The path from staking a claim to piling up gold bars is a long and arduous one. It is one few are equipped to follow, especially if they actually believe they have struck it rich when the claim is staked. Yet the true value is not realized until after the skilled goldsmith has crafted those bars into something worth much more than its weight in gold. In the meantime, our collective glorification of and fascination with so-called invention—coupled with a lack of focus on the processes of prospecting, mining, refining, and adding value to ideas—says to me that the message is simply not having an effect on how we approach things in our academies, governments, or businesses.

Too often, universities try to contain the results of research in the hope of commercially exploiting the resulting intellectual property. Politicians believe that setting up tech-transfer incubators around universities will bring significant economic gains in the short or mid-term. It could happen. So could winning the lottery. I just wouldn't count on it. Instead, perhaps we might focus on developing a more balanced approach to innovation—one where at least as much investment and prestige is accorded to those who focus on the process of refinement and augmentation as to those who came up with the initial creation.

To my mind, at least, those who can shorten the nose by 10% to 20% make at least as great a contribution as those who had the initial idea. And if nothing else, long noses are great for sniffing out those great ideas sitting there neglected, just waiting to be exploited.

Bloomberg Businessweek

Innovation & Design

http://www.businessweek.com/innovate/content/oct2009/id20091021_629186.htm

The Mad Dash Toward Touch Technology

By Bill Buxton October 21, 2009

Buried within the current mad scramble towards touch and multitouch technologies lies an important lesson in innovation: "God is in the details" (Ludwig Mies van der Rohe).

So while executives and marketers all seem to be saying, "It has to have touch," I am more inclined to say that anyone who describes a product as having a "touch interface" is likely unqualified to comment on the topic. The granularity of the description is just too coarse. Everything—including touch—is best for something and worst for something else. True innovators needs to know as much about when, why, and how not to use an otherwise trendy technology, as they do about when to use it. Let me explain.

The photo above shows four watches in my collection. On three of them (a, b, and c), the entire crystal is a touchscreen. Three of them (a, b, and d) have built-in calculators.

When Fat Fingers Meet Small Targets Watch (a) is the Casio AT-550. Despite its conservative styling, it has some pretty amazing software. To put it into calculator mode, you push a button on the lower left side. To enter numbers or operators into the calculator, you just draw them on the crystal with your finger. So, for example, a downward stroke from 12 to 6 o'clock enters the digit one (1), whereas the same stroke followed by a horizontal stroke from 9 to 3 o'clock enters a plus (+) sign. The numbers appear in the main part of the LCD window, and the current operator as a kind of superscript, above them.

The whole screen is used for entering each character, thereby bringing the scale of the action well within the bounds of normal human finger motor control. Less obvious but just as important, the technique enables "heads up" data entry—the equivalent of touch typing. In other words, I can input numbers without diverting my gaze from you or the document from which I am copying a number.

Watch (b) is the Casio TC-50. To put it into calculator mode, you also push a button on the lower left side of the watch. In this case, however, a graphical representation of the familiar calculator numerical keypad appears on the watch face. To enter a number, you touch the desired digit on the virtual keypad. To enter an operator, you touch the appropriate icon
(Ã·, x, -, +) permanently marked just below the LCD at the bottom of the watch crystal. The design is intended to take advantage of your previous experience with calculators. However, while this all seems clear, it does little to make the calculator usable. The watch is a victim of what happens when fat fingers meet small targets—even when accompanied by high concentration. As for touch typing, forget it.

Important Product Lessons Watch (c) is a Tissot Touch. While the crystal is touch-sensitive, this watch does not have a calculator. To activate the touchscreen you push and hold the watch stem for a couple of seconds.

Different functions are enabled by touching the crystal at particular places. For example, if you touch at the 6 o'clock digit, the hands of the watch align and point north, converting the watch into a compass.

Watch (d) is a third calculator watch, a Casio Data Bank 150. This one has a physical, mechanical keypad rather than a touchscreen. While the physical keys are small, they can be accurately used, but not without looking.

What I like about these watches is their power to teach us, using relatively simple existing products, important lessons about products that we might be dreaming about. Take watches (a), (b), and (c). Even though they are all just watches, and all use a touchscreen to gain access to their functionality, knowing how to use any one of them buys you pretty much nothing in terms of knowing how to use the other two. Even if you know how to use two of them, you still don't know how to use the third.

In fact, isn't it interesting to note that there is a closer affinity between the touch interface of (b) and the non-touch interface of (d) than between the two touch ones? In light of this, what in terms of user experience is conveyed by specifying that a product requires a touch interface? Very little. Yet how many of those insisting on a touch interface know about products such as these, much less the lessons that they have to teach?

Touch Isn't New As with almost any suddenly hot technology, touch and multitouch are decidedly not new. They are a textbook example of my notion of the "Long Nose of Innovation. " For example, multitouch was first discovered by researchers in the very early 1980s, before the first generally available PC using a mouse was commercially released. It has been gradually mined and refined ever since. The companies whose products have initiated the current buzz just happened to recognize the latent value of touch, and believe in it enough to take on the risk and investment required to effectively exploit its potential.

Significantly, these companies neither invented the underlying technology, nor were they the first companies to exploit it commercially. This is not a criticism, by the way, but rather a respectful commentary on the nature of design and innovation—one that counters the myth of the genius inventor, and gives appropriate recognition to those who laid the foundation that enabled this to happen.

Understand the Long Nose Finally, consider the following: Casio released the AT-550 in 1984 for under $100. That's the same year that the first Macintosh was released. Working Moore's Law backward, that means that wonderful "heads up" character recognition was created using only one 131,072th of the computer power that would be found on an equivalently sized chip today.

There is a serious lesson here for those would-be innovators who, on seeing the great success of one company's use of some technology or another, scramble to adopt it in the hope that it will bring them a share of that wealth as well. Such behavior is more appropriate for lemmings than innovators.

Rather than marveling at what someone else is delivering today, and then trying to copy it, the true innovators are the ones who understand the long nose, and who know how to prospect below the surface for the insights and understanding that will enable them to leap ahead of the competition, rather than follow them. God is in the details, and the details are sitting there, waiting to be picked up by anyone who has the wit to look for them.

Bill Buxton is Principal Scientist at Microsoft Research and the author of *Sketching User Experiences: Getting the Design Right and the Right Design*. Previously, he was a researcher at Xerox PARC, a professor at the University of Toronto, and Chief Scientist of Alias Research and SGI Inc.

(NASA-CR-194243) THE SENSOR FRAME   N94-70016
GRAPHIC MANIPULATOR Final Report
(Sensor Frame) 27 p

                                            Unclas

                                 Z9/61 0183157

```
The Sensor Frame Graphic Manipulator
      NASA Phase II Final Report
```

PROJECT SUMMARY

PURPOSE OF THE RESEARCH:

Most of the useful information in the real world resides in humans, not in computers. Therefore we must find better ways of moving spatial information *from the human to the computer*. Quality 3-D graphics displays are necessary but *not sufficient* for a highly interactive and intuitive human interface. We need to improve input devices that capture human gestures and spatial knowledge.

One problem associated with direct manipulation interfaces in a design environment is that the user may not be skilled or precise enough to achieve the desired result. We can alleviate this problem through the use of *constrained virtual tools*. We define virtual tools as tools, displayed on the computer's video monitor, which are analogous to the tools used in factories, machine shops, or design studios. They include, but are not limited to, tools for cutting, smoothing, shaping, or joining operations. Virtual tools would map multifinger two and three-space gestures into the operations performed by the "business end" of the tool (such as the blade of a cutting tool), with constraints imposed by the model of the tool itself, the material or workpiece being operated upon, and the objectives of the user. The virtual tool would allow us to sculpt a smooth 3-D surface, varying the curvature or even the smoothness of a curve as it is drawn. However, the manipulation of a virtual tool requires *more* than six degrees of freedom. We believe that optical gesture recognition can provide up to twelve degrees of freedom per hand without the necessity for wires or gloves which inhibit casual use. The essential purpose of our research was to implement the enabling technology which makes casual use of virtual tools possible.

RESEARCH ACTIVITIES:

A prototype Sensor Cube was built using a neon-tube light source for contrast enhancement. A UNIX X-Windows interface was developed, and a control-panel builder was designed and implemented using X-Windows. A gesture-analysis package was developed, and is currently being extended for use in a multiple-finger environment.

RESEARCH RESULTS:

During the course of development of the three-dimensional Sensor Cube, we were informed that the sensors intended for use in the cube would no longer be available (see Section 3.1 for a more detailed discussion). This forced us to evaluate different approaches to optical multifinger sensing. Subsequently, we discovered a method of building the Sensor Cube with only one CCD sensor. This development will allow the three-dimensional Sensor Cube device to be less expensive than it's predecessor, the Sensor Frame. Unfortunately, the need to redesign the optical system and controller hardware and software of the cube delayed completion of this part of the project. Interesting and useful algorithms for 3-D finger tracking were developed and will be evaluated in detail as soon as sensor cube construction and interfacing are complete.

POTENTIAL COMMERCIAL APPLICATIONS:

The two-dimensional Sensor Frame technology will soon be supplanted by the three-dimensional capability of the Sensor Cube. However, the technology developed for use in the Sensor Frame has been transferred to a recently-announced commercial musical-instrument controller, the VideoHarp. The VideoHarp has attracted widespread attention in electronic-music circles, and was recently featured on the cover of Computer Music Journal (Volume 14, No. 1, MIT Press).

Sensor Cube gesture-recognition technology has it's greatest potential impact in computer-aided design (CAD) and teleoperation. Current input devices with six degrees of freedom or less are inappropriate for the manipulation of virtual tools. By gaining additional ability to capture the gestures of skilled scientists, designers, and technicians, computers will become a better alternative to traditional manual methods of design. If desktop manufacturing workstations with gesture-recognition input devices having up to 12 degrees of freedom can do for designers what time-sharing did for the programmers of the punch-card era, human productivity might be enhanced considerably; possibly by orders of magnitude.

1. Background and Motivation For Gesture-Based Systems

1.1. Virtual Reality and Virtual Tools

By the time a human child begins to speak, it has already spent approximately eighteen months to two years learning how to identify objects, people, and actions. It can distinguish one parent from another. It can distinguish itself from other objects and people. It can grasp and manipulate objects. Spatial knowledge comes early, and preceeds language.

Many young children can thread a nut onto a bolt before they go to school. A child less than four years old can do this. The task requires more than six degrees of freedom per hand (ie - positioning and orientation of the object in three-space plus a grasping operation), and implies that manipulation of twelve or more independent parameters is not unusually difficult for a young human.

In contrast, most workstations available today allow simultaneous manipulation of only *two* independent parameters, using a mouse. One *can* specify and manipulate representations of three-space objects with a mouse; but decomposing a six-parameter task into at least three sequential two-parameter tasks is not only counterintuitive, time-consuming, and error-prone; it is a waste of time if we can find a better way. By analogy, we could probably show that anything one can do using a keyboard can also be done using a telegraph key. But most of us would not exchange our computer keyboards for telegraph keys, despite the fact that the latter is cheaper, simpler, smaller, and standardized.

These considerations have prompted several researchers to attempt to improve workstation interfaces with a view toward accommodating human gesturing and tool-manipulation ability. In section 1.3, we will describe several systems which permit manipulation of objects in three dimensions. We will discuss their usefulness and their drawbacks, and ask how they might evolve in the future. While much of the published literature on 3-D input devices concentrates on the videogame-like ambiance of virtual reality, we will move the emphasis toward the idea of virtual tools, a subset of virtual reality that concerns itself with the development of more productive tools for use in design. Design and the need for redesign are among the most costly components in the production of high-technology products such as airplanes, rockets and space vehicles, and of low-tech mass-produced products such as automobiles.

1.2. Virtual Tools

One problem associated with direct manipulation interfaces in a design environment is that the user may not be skilled or precise enough to achieve the desired result. We can alleviate this problem through the use of *virtual tools*. We define virtual tools as tools, displayed on a workstation's video monitor, which are analogous to the tools used in factories, machine shops, or design studios. They include, but are not limited to, tools for cutting, smoothing, shaping, or joining operations. Virtual tools would map multifinger two and three-space gestures into the operations performed by the "business end" of the tool (such as the blade of a cutting tool), with constraints imposed by the model of the tool itself, the material or workpiece being operated upon, and the objectives of the user. The virtual tool would allow us to sculpt a smooth 3-D surface, varying the curvature or even the smoothness of a curve as it is drawn.

Virtual tools might be used to add material to a workpiece, to cut material, or to extrude it. The motion of a tool might be low-pass filtered, with filter-cutoff frequency of the filter being controlled,for example, by the distance between two fingers.

As we evolve hierarchies of virtual tools, designer productivity will hopefully increase. If we can significantly shorten design time, customization will be easier... and it is important to realize in this context that the higher-order goods of mass production, the machines that make other machines, are often highly customized tools, made in small quantities, but requiring many design iterations over their useful lifetime. As we build the virtual tools that cut design time, learning time for the designer will also be shorter, in relation to productivity. This is especially true if the designer can see "immediate feedback" on his or her latest design at low cost.

1.3. Related Research In Gesture-Sensing Technology

How can we best capture human gestures for intuitive manipulation of spatial objects? There are several different approaches to solving this problem. First, let's look at several currently-available devices:

* The DataGlove (VPL Systems)

* The Dexterous Hand Master (Exos)

* The Spaceball (Spatial Systems)

* The Flying Mouse (SimGraphics Engineering Corp.)

The DataGlove and Dexterous Hand Master (DHM) both sense finger-flexing motions. The DataGlove also senses hand position and orientation using a "Polhemus sensor" developed by McDonell-Douglas. The Polhemus sensor determines position and orientation of the hand using an externally-generated oscillating electromagnetic field. The version of the DataGlove with a Polhemus sensor has the advantage that it can sense relatively large-scale hand positions and orientations. Knowing position and orientation of the palm of the hand, one can use knowledge of finger-joint flexure to determine fingertip position, for use in grasping and tool-manipulation applications. In addition, by inserting piezoelectric transducers in the fingertips of the glove, one could conceivably provide some degree of touch feedback. Force feedback is a more difficult problem. The DHM has the advantage that its determination of finger-joint flexure appears to be considerably more accurate and repeatable than that of production DataGloves. It has the disadvantage that it does not currently provide hand position and orientation, although this could probably be implemented if market demand warrants it. Users of the DHM assert that it is lighter and less encumbering than it looks, although the time required to fit it to the hand seems to preclude casual use.

The use of glove-like sensors to sense gestures poses some problems. Currently, these devices use a cable to transmit data from the glove to the workstation, making casual use difficult. More later about the importance of casual use.

Hand (and consequently fingertip) position sensing (as opposed to detection of finger-joint flexure) requires the use of the relatively-expensive Polhemus sensor, and its use can be complicated by the presence and movement of ferrous metals in the vicinity of the sensor. A variation of the DataGlove developed by for Nintendo games, the PowerGlove, uses sonar devices mounted in the glove, but this severely constrains the orientation of the hand.

In the case of the DataGlove, unless each user has his own glove, a workstation supporting the device must have multiple gloves available in order to support left and right-handed persons with varying hand sizes. The same is probably true for the Exos device. Neither device yet provides sufficiently accurate and repeatable fingertip position information for use in a virtual tool environment. These latter considerations are an argument against the use of glove-like devices in a virtual tool (as opposed to virtual-reality) environment. Nevertheless, for many applications, we should expect them to provide a reasonably cost-effective solution.

The Spaceball is essentially a 3-D joystick. It is a ball slightly larger than a tennis ball, mounted in such a way as to make extended use very comfortable. The spaceball is excellent for positioning and orienting displayed 3-D objects, and for modifying one's view of a stationary object. It has good accuracy and repeatability. Because it functions like a joystick, it has some of the disadvantages that the joystick has relative to a mouse, and it has only six degrees of freedom. Six degrees of freedom are adequate for positioning and orienting objects, but more degrees of freedom are required to manipulate virtual tools. Once the tool is positioned, there are more things we must do to make it work, and that is the problem.

The Flying Mouse is a three-button mouse with a Polhemus sensor inside, designed so that it is easy to pick up. One can position and orient it in space, and then press the buttons. This is almost good enough for virtual tools, but not quite. For virtual tools, one might prefer the buttons to be more analog, ie - pressure sensitive. A nice thing about the Flying Mouse is that it can function as a normal 2-D mouse when on a tabletop, a convenient feature. The builder, Simgraphics Engineering Corporation, is well aware of the importance of the design and CAD markets, and emphasizes development of software necessary for the future "virtual tool" environment.

It is important to point out that the technologies we are describing are in their infancy, and constantly evolving. For this reason, many of the remarks pertaining to the products described above may become quickly outdated.

1.4. The Next Step: Vision-Based Gesture Sensing

The devices described in section 1.4 generally involve the use of mechanical, magnetic, or Hall-effect sensors in the sensing of palm position or finger flexure. A different approach to the problem of sensing multifinger gestures involves the use of vision-based systems.

Computer vision systems that analyze complex real-world scenes in real time remain beyond the state of the art. Nevertheless, in some applications, such as visual inspection, where scenes are specialized and predictable, systems are approaching feasibility (and a few systems are in commercial use).

At Sensor Frame Corporation in Pittsburgh, we have developed a device called a *Sensor Frame*, a 2-D optical finger-tracking device developed by the author and colleagues at Sensor Frame Corporation and Carnegie Mellon University. The prototype Sensor frame, using four sensors, reliably tracks up to three fingers at 30 Hz despite the fact that fingers sometimes block one-another from the point-of-view of some of the sensors. Tracking of *multiple* fingers is what distinguishes it from commonly-available touch screens. A drawing of the Sensor Frame, mounted on a monitor and in "standalone" mode, is shown below. The Videotape accompanying this report as Appendix C-1 shows the Sensor Frame in use.
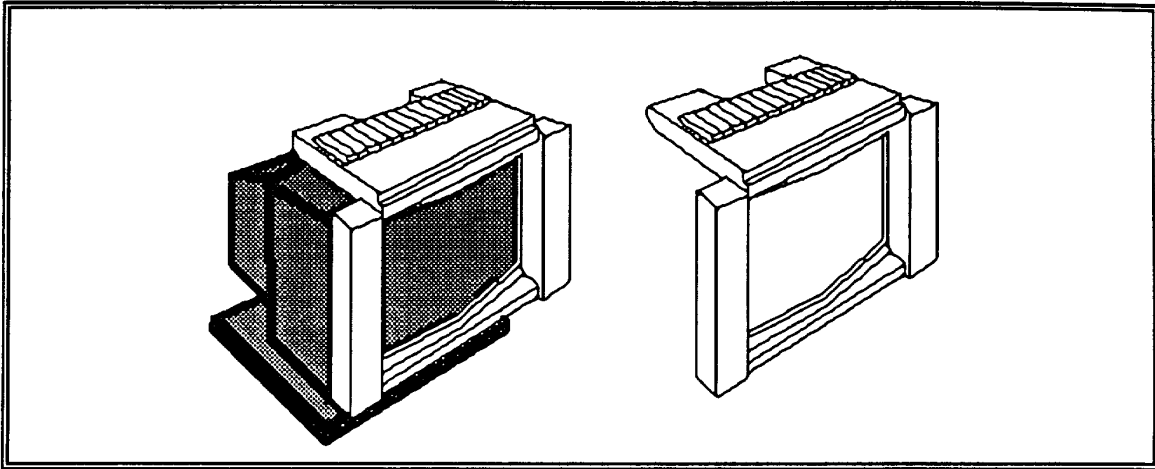
The Mark IV Sensor Frame

Although the Sensor Frame represents a technology still in the early stages of it's development, it has aroused a fair amount of interest in industry, the press and media. In late 1988, CNN featured the Sensor Frame and VideoHarp in their AT&T Science and Technology series, and in 1989 Business Week featured both devices in their technology section. The Sensor Frame also appeared on the cover of NASA Tech Briefs, together with a feature article.

Unfortunately, production of the Sensor Frame, intended for September of 1989, was abruptly halted when the sensor manufacture halted delivery of optical dynamic-RAM sensors in the spring of 1989. This development is discussed in more detail in section 3.1. At present, we are developing the Sensor Cube, a 3D extension of the Sensor Frame, which will use one area CCD sensor to track up to three fingertips in three dimensions.

1.5. Future Applications of Gesture-Based Systems

Much of the motivation for building gesture-based systems can come from thinking about how we might apply them in the future in order to increase the productivity of designers. When we ask which gesture-sensing input devices will survive, we need to ask what future applications will require. Let's do a little thought experiment, and imagine what we would like our workstation to do for us if our objective were to design or modify a three-dimensional object, such as a machine-tool part, a piece of furniture, a molecule, or a nozzle for a rocket engine. We'll call this new type of workstation a *desktop manufacturing (DTM) workstation*, because it is intended to permit rapid prototyping of real-world objects. It would enable a designer to interactively specify or modify the shape of an object using spatial gestures and the *virtual tools* described above. Then it would build the object.

The DTM workstation would consist of the following components:

* A powerful CAD workstation that displays colored, shaded 3D objects, with full-motion video capability.

* A "3-D copier" similar to the stereolithography device manufactured by 3-D Systems Corporation. This device, or some future variation of it, will be used to fabricate a prototype or custom part quickly. There are currently at least three companies working on this aspect of DTM technology, and the number will probably increase.

* A 3-D gesture sensor, with gesture-recognition software and a *virtual-toolmaker's toolkit*.

* An optional 3-D laser scanner for scanning 3-D shapes.

2. Phase II Technical Objectives

Phase II Technical objectives consisted of the following:

1. Development of Sensor Cube hardware and finger-tracking software.

2. Development of an intuitive interface for graphic-object manipulation.

3. Development of X-Window interface and UNIX device drivers for the Sensor Cube.

4. Development of soft control panels.

These objectives correspond to objectives 3.1.1 through 3.1.4, as described in our Phase II proposal for this project. Due to the sudden unavailability of DRAM sensors, as described in section 3.1 of this report, not all objectives were achieved in the form originally anticipated in the Statement of Work. Because the Sensor Cube design had to be modified significantly as a consequence of the sensor-availability problem, the resultant implementation delay precluded implementation of the 3D aspects of task 3.1.2.

3. Methodology, Observations, And Results

3.1. Development of Sensor Cube Hardware

In hardware terms, the *Sensor Cube* described in our NASA Phase II proposal was intended to be a thicker version of the Sensor Frame. A Sensor Cube was built with a 4.5" deep neon light source and four dynamic RAM (DRAM) sensors of the type used in the original Sensor Frame. This first Sensor Cube hardware was completed on schedule, about six months after the inception of Phase II. The first Sensor Cube prototype is shown schematically below, and in a videotape enclosed as Appendix C-2 of this report.

View From Sensor A   View From Sensor B
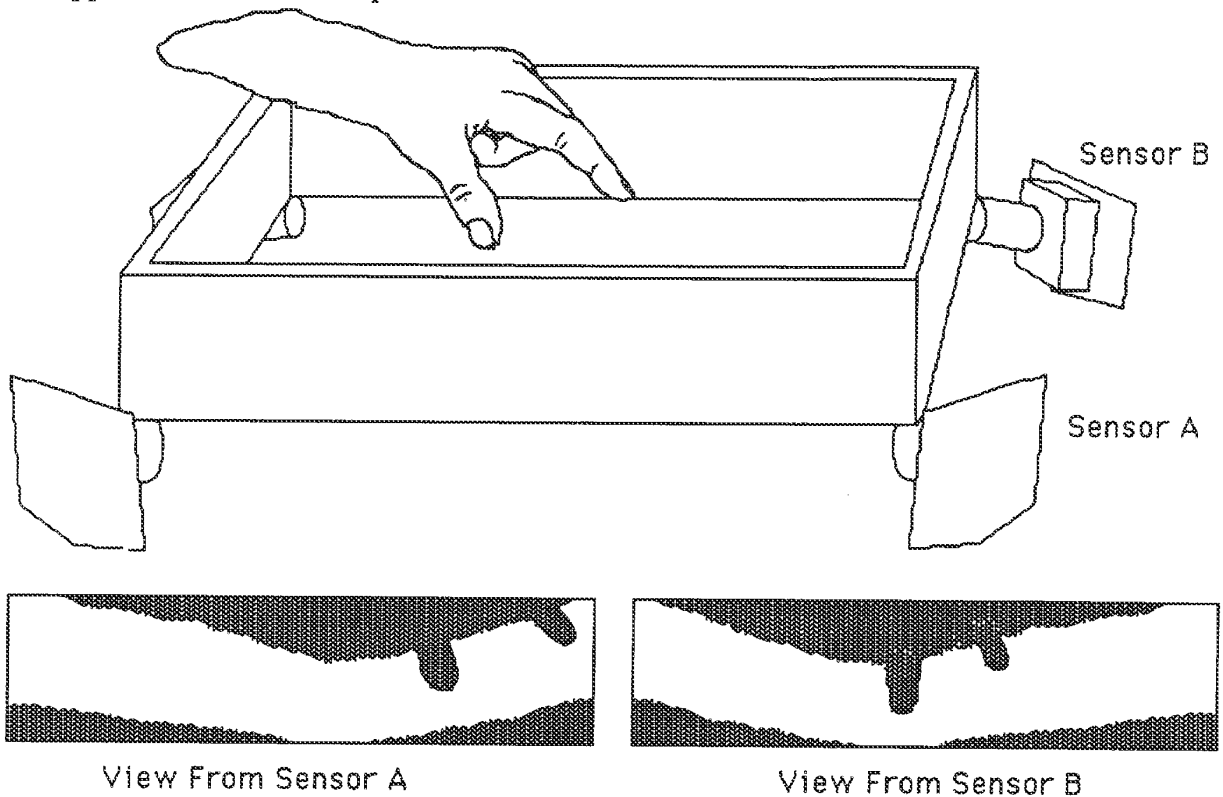
The First Prototype Sensor Cube, and Two Views From the Sensors

After completion of the first Sensor Cube prototype, work began on a UNIX interface for a Silicon-Graphics workstation, and at the same time for an X-Windows interface for an IBM RT workstation.

The UNIX and X-Window projects were essentially complete, in March of 1989, when Sensor Frame Corporation was abruptly informed by Micron Technology Corporation, the sole supplier of the DRAM sensors, that the fabrication of their line of DRAM sensors had been terminated. Our plans for commercial production of the Sensor Frame, intended to begin in August 1989, had to be abandoned. Both the then-current Sensor Frame and Sensor Cube designs made use of the 256K optical DRAMS supplied by Micron. All supplies of the 256K DRAMS had been committed to larger users by Micron before we were informed of the decision, leaving us with only five sensors; enough for our single prototype Sensor Frame, plus one spare. We were told that we would be able to obtain 100 of the smaller 64K DRAMs; however, we considered the 64K devices unsuitable for use either in a commercial Sensor Frame or in a Sensor Cube. Nevertheless, we bought the 100 64K devices, because we had a third product on the drawing boards that *could* use it; the VideoHarp.

It is perhaps relevant at this point to discuss the original reasons for the selection of DRAM sensors rather than charge-coupled devices (CCDs) as sensors, as well as the decision not to seek out another DRAM vendor to supply the optical DRAMs.

In 1982, when the first precursor of the Sensor Frame was built, CCDs were extremely expensive compared to DRAMs, with linear CCDs running in the thousand-dollar range. Further, CCDs require much more complex interface circuitry than do dynamic RAMs. In the early 80's, there were no integrated-circuit devices to provide the complex clock pulses, with their carefully-controlled slew rates, required by CCDs. Although integrated CCD clock and level-conversion chips became available in the mid-to-late 80's, the system cost of reasonable-quality CCDs is still considerably greater than the cost of optical DRAM chips. Further, the DRAM chips had several desirable properties that CCDs currently lack, one of the most important being addressability. In addition, it has not been any easier to obtain a second-sourced CCD than is was to obtain a second-sourced optical DRAM.

Although desperate, we were unable to convince Micron Technology to reverse their decision. They had little incentive to persue this still-relatively-small sensor market, having been awarded a virtual monopoly on the American DRAM market (along with Texas Instruments and IBM, the only remaining American DRAM manufacturers) by the U.S. Department of Commerce decision in 1988 to severely limit the importation of DRAMS from Japan.

As a consequence of this unfortunate event we did two things. First, since we could not manufacture Sensor Frames or Sensor Cubes, we decided to produce the VideoHarp, an optically-scanned musical instrument, which was the only one of the three products resulting from Sensor Frame technology that could make use of the available 64K DRAMs. Second, since we knew that we must switch to a different sensor technology after the 100th VideoHarp was built, and in order to build a commercial Sensor Cube (at present, we believe that it may be possible for a future VideoHarp and Sensor Cube to use the same area sensor), one of us (Paul McAvinney) attempted to find a way to build a Sensor Cube using fewer sensors. This effort succeeded shortly thereafter in the summer of 1989, when we developed a design using only one sensor and two mirrors.

The illustration below shows the a perspective view of the resultant single-sensor Sensor Cube mounted on a video monitor. Unlike the Sensor Frame, this design requires use of a gray-scale sensor such as a CCD or MOS area sensor. However, because it requires fewer sensors and associated optics, it will probably be cheaper to produce than the first Sensor Cube design. It's Z-axis depth will be about six inches, a significant improvement over the original design. In addition, the new design lends itself more easily to use as a two-handed teleoperation device. If two cubes are positioned side-by-side, they can share a controller.
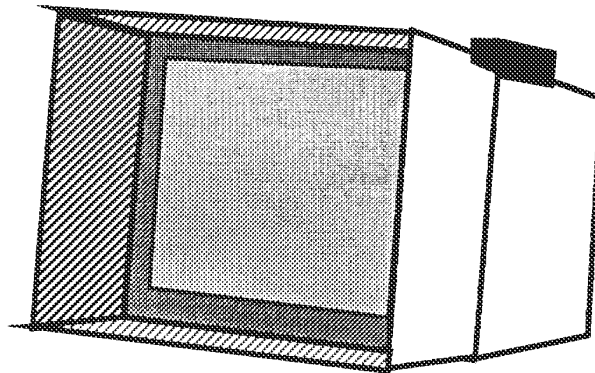
Figure 2: The Prototype Sensor Cube Mounted on a Video Monitor

Several important design considerations are driving the design of the second Sensor Cube. We list here the most important ones:

* The device must allow for at least ten degrees of freedom per hand, hopefully more. This will allow positioning and orientation of a virtual tool relative to a workpiece, followed by x-y manipulation of analog inputs on the tool itself by two opposed fingers. Even twelve degrees of freedom may not be too difficult to obtain.

* The device should allow casual use. This becomes especially important as increasingly powerful virtual tools permit a given operation to be completed in a short time, allowing the user to do something else which may not require the use of the gesture-sensing device. Good virtual tools should preclude the need for constant use, lessening concern about operator fatigue caused by holding one's hand in the air all day.

* The user's hands should be left free to use other devices, such as keyboards and telephones.

* Position of fingers relative to screen objects should be sensed.

* The device should be able to sense fingers in the vicinity of a video monitor. It should be attachable to the monitor, so that the user need not sacrifice desk space.

* It should operate independently of the video monitor, so that it can be mounted in another location (possibly for teleoperation-oriented applications) if the user so desires.

* It should be inexpensive in mass production, in order to encourage general use and standardization of application and user-interface software.

The next illustration shows a perspective schematic view of the new Sensor Cube design. The sensor is at lower left, and the shaded areas represent two mirrors at right angles.
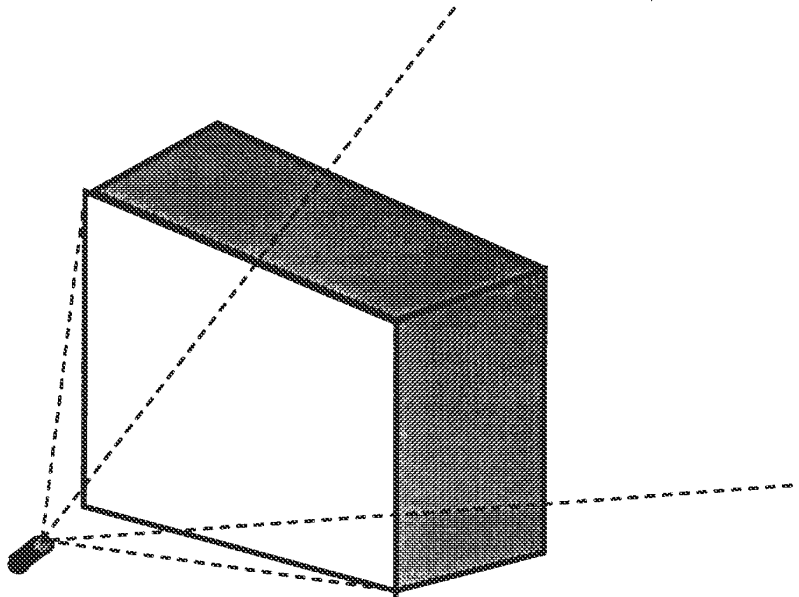
Perspective View of Sensor Cube, With Monitor Screen At Rear

The next illustration shows the Sensor Cube from the front. Also shown are the positions of the *virtual sensors*. The scene produced in the single real sensor, at lower left, includes the scenes reflected from the mirrors along the top and right walls of the Sensor Cube enclosure. These virtual images may be treated geometrically as if they were images seen by the virtual sensors in the three positions shown. The net effect of the mirror system is to provide an image from four directions instead of just one. Since all sensors, real and virtual, look at the hand from a position near the plane of the video monitor, occlusion of fingers by the palm is minimized, except in the cases of extreme rotation of the hand.
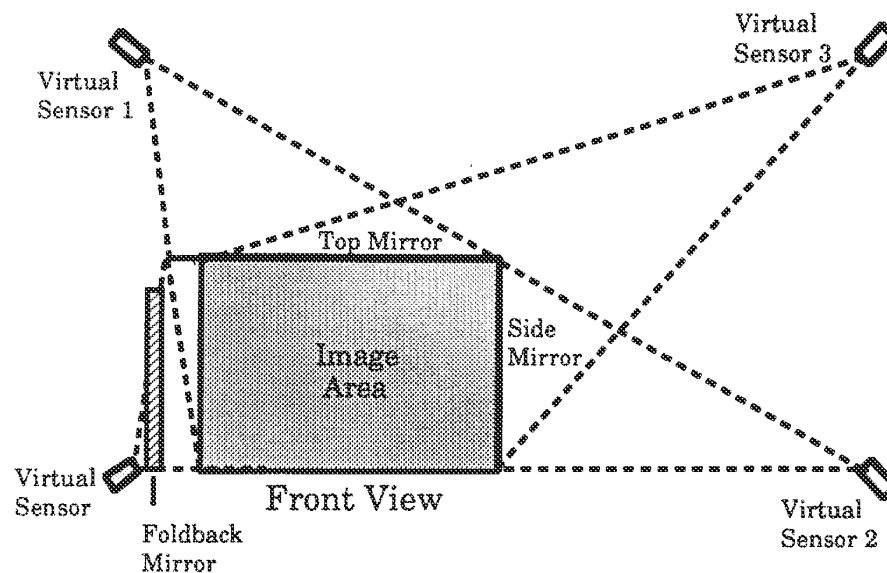
Front View of Sensor Cube, Showing Virtual Sensors

The new Sensor Cube controller is currently under construction. Delays in delivery of support chips for the new design, based on a relatively inexpensive CCD designed by Texas Instruments, preclude the possibility of completion before the end of the NASA Phase II contract. Most U.S.- based CCD vendors have their CCD chips and support circuitry (and associated data sheets) produced in Japan for use in Japanese video cameras, and the U.S. wholesale market is small. As a consequence, some parts that have been on order for six months are still not being delivered.

A more long-term solution to the problem caused by the fact that there are currently no multiply-sourced area image sensors suitable for our designs is for us to design our own area sensor chip. This effort would make use of a scaleable CMOS process and the multi-foundry capabilities of the MOSIS prototyping service offered by the Information Sciences Institute at the University of Southern California (USC/ISI). PC-based software for MOSIS project-chip designs is available from commercial vendors at nominal cost.

Because of the uncertainty in the design schedule for this approach and our limited resources, we chose the more conservative approach of using commercial CCDs. Nevertheless, in the fall of 1989 we submitted a proposal to DARPA to fund a "smart" addressable MOS image sensor chip for use in gesture-based systems, but the proposal was rejected. We were told by DARPA that the proposal was considered technically sound, but that most if not all of their new funding had been reserved for HDTV and Star Wars projects. DARPA's approach may change, given the recent high-level shake-ups within the organization, but Sensor Frame Corporation intends to stake it's future on commercial product development (ie. - the VideoHarp), and fund new sensor development internally.

3.2. The Sensor Cube Finger-Tracking Algorithm

The algorithm for determining the spatial position and orientation of fingers in the Sensor Cube image area, stated here in somewhat oversimplified form, works as follows:

1) From the point-of-view of virtual sensor 3, the furthest sensor away from any sensed object, the scan line which intersects the mirrors at the greatest angle relative to the base-plane is read. This is guaranteed to sense a finger and allow it to be tracked at a z-axis value at or beyond the maximum guaranteed z-axis (Z_{MAX}) tracking value.

2) As any finger approaches Z_{MAX}, it is scanned by a "crosshair" pattern for each virtual sensor. One line of the crosshair is oriented *along the axis of the finger*, the angle being determined from previous scans. This is called the "longitudinal scan". For the simple case of a finger pointing directly along the Z axis, this value, taken from each virtual sensor, determines the position of the fingertip in the Z dimension. Information regarding fingertip position from each *longitudinal* scan is used to determine the height (above the fingertip) of the next *lateral* scan (see below).

3) The second scan is at right angles to the first, scanning across the *width* of the finger. This is called the "lateral scan". Information from each lateral scan is used to determine the lateral position of the next *longitudinal* scan.

In this method of tracking, each longitudinal scan corrects the position of the next lateral scan for a given finger, and vice-versa. Whether a frame-buffered image or an addressable sensor is used, the method allows us to locate fingers by scanning a relatively small fraction of the total number of pixels in the image, greatly reducing Sensor Cube controller processing requirements. In practice, two lateral scans of each finger may be needed to determine finger orientation accurately. When partial occlusion of a finger occurs, things become somewhat more complex. Experience with the Sensor Frame leads us to predict that we should not try to track more than three fingers at a time. This necessitates a style of gesturing which requires folding of the two smallest fingers into the palm. However, such a constraint appears to be easily learnable by most users. Further, the plane formed by three fingertips is useful for determining the orientation of a displayed object "grasped" by the hand. In the future, with more experience, we may try to relax the "three-finger" constraint.

3.3. Development of an Intuitive Interface for Graphic-Object Manipulation

Because the development of the Sensor Cube was delayed, the translation, rotation, grasping, and scaling of graphic objects in three dimensions was not possible. However, Appendix C-1, in the videotape attached to this report, shows how these capabilities were implemented using the Sensor Frame prototype for the two-dimensional case. We believe that when the Sensor Cube becomes operational, extension of these capabilities to the 3D case will not be difficult.

3.4. Development of an X-Window Interface and UNIX Device Drivers for the Sensor Cube.

X-Window and UNIX device-driver interfaces were successfully implemented for the Sensor Frame on IBM-RT and Silicon-Graphics IRIS workstations. The videotapes attached as appendices to this report show the effects of this implementation. Appendix B lists the implemented UNIX device-driver functions written in C.

In general, it was found that the X-Windows interfaces (particularly on the IBM RT) were quite slow due to the excessive overhead of message passing between various X-Window components. This made multifinger tracking and screen update slow and difficult. The widely-acknowledged problem of excessive message-passing overhead has resulted in the recent appearance of terminals with processors dedicated to the efficient execution of X Windows.

Our implementation of the Sensor Frame on the Silicon-Graphics IRIS workstation was done using Sun's NEWS windowing system provided by Silicon Graphics.

3.5. Development of Soft Control Panels

The control-panel editing program was developed by researchers at Carnegie Mellon University under subcontract to Sensor Frame Corporation. An article describing this effort, "A Gesture Based User Interface Prototyping System", by Dr. Roger Dannenberg and Dale Amon of the School of Computer Science at Carnegie Mellon, was published in the Proceedings of the Second Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, November 1989. That article is included in it's entirety as Appendix A of this report. The attached videotape (Appendix C-3) shows the operation of this system.
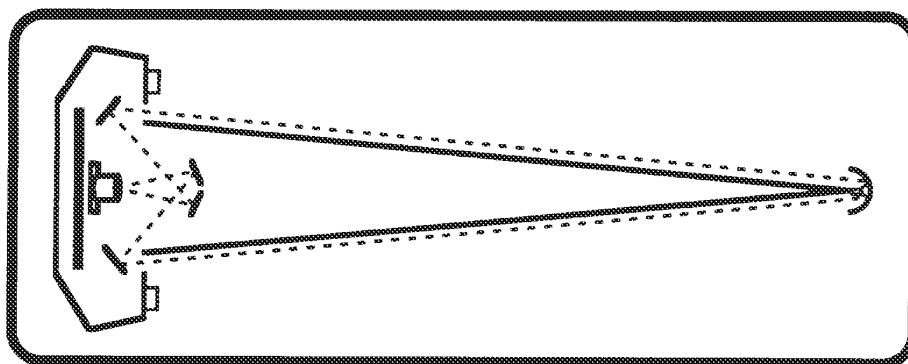
3.6. The VideoHarp
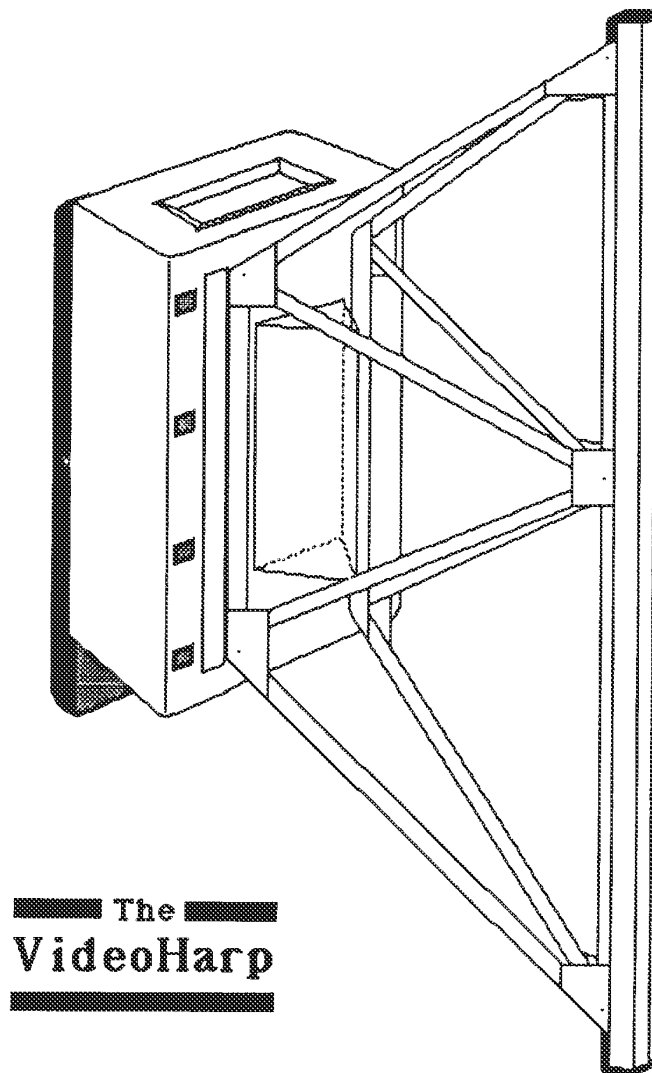
The VideoHarp is an optically scanned musical instrument which converts moving images of the fingers into music. Unlike keyboards and other mechanical sensing devices, the VideoHarp, because of the flexibility of it's optical scanning method, can recognize many classes of musical gestures, including bowing, strumming, keyboarding, and even conducting. A given class of gesture may be applied to any class of instrument timbre. For example, one could bow a horn or strum an organ. Using a fixed mechanical controller, such as a keyboard, one could produce non-keyboard sounds, such as the sound of a stringed instrument. However, even a keyboard with aftertouch cannot vary the timbre of a bowed note significantly as the note is played. Note that a cellist or violinist can press harder on the bow, play closer to the bridge of the instrument, and produce vibrato all at the same time. Keyboard controllers do not permit such quantitative, intuitive, and flexible control of many parameters at once.

The VideoHarp, because it can optically track all the player's fingers at once, allows control of many independent parameters. It permits a richness of timbral expression approaching, and often exceeding, that of traditional instruments. The playing surfaces of the VideoHarp can be divided up into *regions*. Each region possesses it's own attributes, such as which instrument is to be played, the width of keys, pitch and amplitude ranges, and many others too numerous to mention here.[1]

Some classes of gestures lend themselves well to *conducting*. For example, a stored score can be conducted using bowing motions. Each reversal of the bow causes the next note to be played. While one hand executes the bowing motions, the fingers of the other hand can be used to control additional aspects of timbre at the orchestral level[2]. This allows a novice user to obtain immediate musical results, and to express himself musically at a high level without having to learn all the nuances of the instrument. It is the *musical expression* which is important here, not the ability to specify which notes are to be played. That has already been done by the composer. Although a conductor may look at an orchestral score in order to plan what to do next, he is primarily interested in developing his own individualized expression or interpretation of the composition.

The following diagram illustrates the internal structure of the VideoHarp, as seen from above. The dashed line shows the light path from the light source (at right) to the sensor, at left. Mirrors are used to bend the light path so that both playing surfaces can be scanned by a single area sensor. Fingers placed against the playing surface block light from the light source, creating a shadow image on the sensor after being focused by a lens system (the cylindrical object at left).

The V2 VideoHarp, As Seen From Above

The VideoHarp can assume four different roles:

• **A Musical-Instrument Controller:** The VideoHarp is an optically-scanned free-hand gesture sensor adapted to the needs of the instrumentalist. It can be connected to any synthesizer with a MIDI input.

---

[1] For more information, see *The VideoHarp*, in Proceedings of the 14th International Computer Music Conference, Cologne Germany, 1988, Ed. Lishka and Fritsch.

[2] An orchestra can be thought of as a large instrument played by a conductor. The conductor does not specify the notes to be played, only *how* they are to be played.

• **A Conducting Controller:** The VideoHarp can capture gestures used in *conducting* a group of instruments, such as a quartet, an ensemble, or even a full orchestra.

• **A Composition Tool:** With it's ability to optically sense playing and conducting gestures of many types, the VideoHarp is an *enabling technology* which permits composers to experiment with the interaction between melody, tempo, timbre, and dynamics, with a flexibility and immediacy unmatched by current controllers.

• **A Complete Musical Instrument:** In the future, a VideoHarp with built-in synthesizer will be a complete musical instrument. At present, because there is no "standard" synthesizer, it is better to leave the choice of this device up to the user.

The VideoHarp has received national and international coverage in several publications, including Science News and Business Week. A color picture of the VideoHarp appeared recently on the cover of Computer Music Journal, which included a paper by the inventors.

4. Conclusions And Recommendations

We beleive that in the long run, vision-based gesture recognition systems such as the Sensor Cube will be widely used; first in design workstations, and later in personal computers, when full-motion video display of virtual tools and workpieces becomes inexpensive (this may happen relatively soon). We believe this for the following reasons:

- Casual, hands-free use of virtual tools will become increasingly important to users as the number, quality, cost, and utility of constrained virtual tools continues to shorten the design process and increase the number of people who will make use of it.

- Desktop Manufacturing (DTM) will allow fast prototyping, quick redesign, and inexpensive small-batch production of evolving products. As DTM becomes cheaper, a wider base of users will insist on standardized and portable virtual tools.

- Because each virtual tool must contain a description of the gesture-to-toolblade mapping, optical, rather than mechanical methods of gesture sensing permit the most flexible and repeatable interpretation of gestures having on the order of twelve degrees of freedom from a wide range of human hand and finger shapes.

- The Sensor Cube will be inexpensive in large quantities, and unobtrusive in casual use.

In the short run, we have to survive; we have had our problems obtaining a reliable supply of appropriate sensors and support circuits in a sensor market still dominated by video cameras for television applications. This situation has delayed construction of the Sensor Cube prototype (see Section 3.1), but things will probably improve. One Japanese image-sensor manufacturer has already requested that we submit a detailed proposal to them outlining our design requirements for a "smart" addressable area sensor. American IC manufacturers continue to lag in their understanding of the future role and importance of smart optical sensors which can detect and flag the pixel locations of image changes in the time domain.

We believe that the next great revolution in human productivity will be the result of a nonlinear increase in the utility and productivity of design tools. Good tools will make design more fun, and human creativity and productivity always profit when a process is viewed as being fun rather than work.

Appendix A: Reprint of Dannenberg/Amon SIGGRAPH Article

A Gesture Based User Interface Prototyping System

Roger B. Dannenberg and Dale Amon

School of Computer Science
Carnegie Mellon University
email: Roger.Dannenberg@cs.cmu.edu

Abstract

GID, for Gestural Interface Designer, is an experimental system for prototyping gesture-based user interfaces. GID structures an interface as a collection of "controls": objects that maintain an image on the display and respond to input from pointing and gesture-sensing devices. GID includes an editor for arranging controls on the screen and saving screen layouts to a file. Once an interface is created, GID provides mechanisms for routing input to the appropriate destination objects even when input arrives in parallel from several devices. GID also provides low level feature extraction and gesture representation primitives to assist in parsing gestures.

1. Introduction

Gestures, which can be defined as stylized motions that convey meaning, are used every day in a variety of tasks ranging from expressing our emotions to adjusting volume controls. Gestures are a promising approach to human-computer interaction because they often allow several parameters to be controlled simultaneously in an intuitive fashion. Gestures also combine the specification of operators, operands, and qualifiers into a single motion. For example, a single gesture might indicate "grab this assembly and move it to here, rotating it this much." Previous work on gesture based systems [1, 2, 6, 4, 12] has only begun to explore the potential of gestural input. We need a better understanding of how to construct gestural interfaces, and we need systems that allow us to prototype them rapidly in order to learn how to take advantage of gestures. Our work is a step toward these goals.

Building interactive systems based on gesture recognition is not a simple task. As we designed and implemented our system, we encountered several problems which do not arise in more conventional mouse-based systems. One problem is supporting multiple input devices, each of

which might have many degrees of freedom. Unlike most mouse-based systems which can only engage in one interaction at a time, our system supports, for example, turning a knob and flipping a switch simultaneously.

Another problem is how to parse input into recognized gestures. We assume that gestures are specific to various interactive objects. For example, a switch displays an image of a toggle on the screen and can be "flipped" by a fingertip, but only if the finger travels across the image in the right direction. In this case, finger motion must be interpreted in the context of the interactive object, and a path (as opposed to instantaneous positions) defines the gesture.

Beyond these problems, we were also interested in making our prototyping environment easy to use, modular and extensible. Thus, we have been concerned with the issues of how to combine interactive objects in a screen-based interface, how to edit the layout and appearance of the interface, and how to encapsulate the behaviors of interactive objects and isolate them from other aspects of the system.

A final issue is the question of debugging support to aid in the implementation of new interactive objects. We use input logging to make bugs more reproducible and a combination of interpreted and compiled code to speed development.

We have completed a system, named GID for Gestural Interface Designer, in which one can interactively create and position instances of interactive objects such as menus, knobs and switches. One can interactively attach semantic actions to these objects. GID supports input from both a mouse and a free-hand sensor that can track multiple fingers. We are far from having the ultimate gesture based interface support environment, but we have developed interesting new techniques that are applicable to future gesture-based systems.

In section 2 we describe the structure of our prototyping system, and section 3 describes the handling of input from multiple devices. In section 4 we describe our general technique for processing input in order to recognize gestures. Section 5 describes in greater detail our develop-

ment techniques and the current implementation. Conclusions are presented in section 6 along with suggestions for future work.

2. The Interface Designer

This project extends an earlier effort called Interface Designer, or ID. The goal of ID was to provide a small, practical and portable system for creating screen-based interfaces by direct manipulation. ID was inspired by Jean-Marie Hullot's work at INRIA, a precursor to Interface Builder [5, 9]. A typical use of ID might be the following: by selecting a menu item, the user creates an instance of an object which displays a 3-D database. In order to manipulate the image, the user creates a few instances of sliders. A short Lisp expression is typed to supply an action for each slider, and labels of "azimuth", "altitude" and "pitch" are entered. Now, moving a slider causes a message to be sent to the display object and the image is updated accordingly.

The basic internal structure of ID introduces no significant improvements over other object-oriented event-driven interface systems such as MacApp [11] or Cardelli's user interface system [3]. It will be described here, however, for clarity.

ID represents the screen as a tree of objects. At the root is a screen object that contains a set of window objects. Each window object may contain a set of control objects. One type of control object is the control group, which serves to collect a set of control objects into an aggregate. Other types of control objects include sliders, buttons and switches of various styles. (See figure 2-1.)
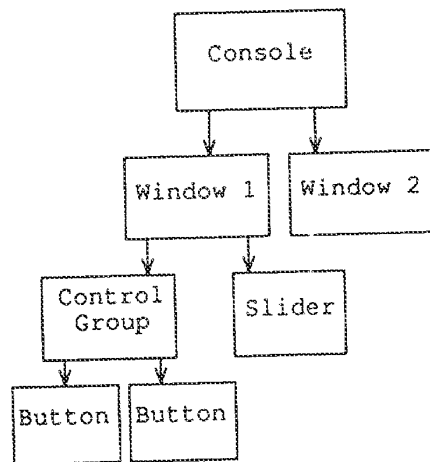

Figure 2-1: An ID control object tree.

In addition to the hierarchy implied by this tree, there is also a class hierarchy arranged so that classes can inherit much of their behavior. (See figure 2-2.) The Input-Control class encapsulates generic behavior of objects that handle input from the user and manage some sort of image

on the screen. PictureControls, a subclass of Input-Controls, actually draw images. These include classes such as Switch and Slider. Another subclass of Input-Control is ControlGroup, which implements the search for an input handler. New interactive controls are typically created by subclassing PictureControl or one of its subclasses. Output-only "controls" have also been defined as subclasses of Control. For example, class 3dPict draws a wire-frame rendering of a 3-D data base which is loaded from a file.

```
Object
   Control
      InputControl
         PictureControl
            Button
            Switch
            Slider
            FontDev
         ControlGroup
            Console
            Window
            Menu
            MenuCard
            PictureGroup
         MenuItem
      OutputOnlyControl
```
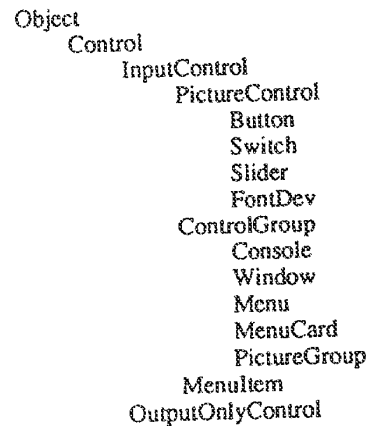
Figure 2-2: Interface Designer class hierarchy.

In normal operation, ID has a single main loop that waits for input and delivers it to the appropriate destination. Each input event is represented by a window identifier, a device type (e.g. mouse or keyboard), coordinates (if any), and other data. This event is passed to the root of the tree where a search for a recipient begins. Typically, each node which is not a leaf node (a PictureControl) passes the event to each of its children until one of them accepts the input event.

To make this recursive search reasonably efficient, a ControlGroup object rejects mouse input which falls outside of its bounding box, and windows reject input unless the event's window identifier matches. Even with these optimizations, it is too inefficient to search the object tree from the root for each mouse-moved event during a dragging operation. Instead, a context mechanism is used.

In ID, the handler for input is found at the top of a *context stack*. An object can grab future input events by pushing a new context onto the stack to direct future input to the object. For example, a dragging operation would start with a mouse-down event that would be handled in the normal way. Upon receiving the mouse-down event, the object that handles the dragging operation pushes the context stack and becomes the target of future input. All successive mouse-move events go directly to the object. When mouse-up is received, the object pops the current context to restore input processing to normal.

The context stack has two uses in addition to temporarily grabbing mouse input. The context stack is used for nested

128

pop-up windows and also for implementing an "edit" mode in which control objects can be created, moved, copied, and deleted. In edit mode, we want to be able to select controls without invoking their normal operations. This is accomplished by pushing a special "edit context" which routes all input to an editor that can manipulate the on-screen objects.

3. Parallel Input Handling
We used ID as the basis for GID, our gesture-based system. GID was designed to be used with a Sensor Frame [7] as the gesture sensing device. The Sensor Frame tracks multiple objects (normally fingers) in a plane positioned just above the face of a CRT display. The "plane" actually has some thickness, so three coordinates are used to locate each visible finger. When a finger enters the field of view, it is assigned a unique identifier called the *finger identifier* Each time the finger moves, the new coordinates of the finger and the finger identifier are transmitted from the Sensor Frame to the host computer. Ideally, when a finger enters the field of view of the Sensor Frame, it is assigned a number which it retains for the entire time it remains in view. Since the Sensor Frame may be tracking multiple fingers in parallel, coordinate changes for several fingers may be interleaved in time.

In our gesture-based system, we wanted be able to handle multiple finger gestures acting on a single object, for example, turning a knob. We also wanted to allow users to operate a control with each hand. The stack-based context mechanism described in the previous section, however, does not allow inputs to be directed to several objects. We could simply pass all input to the root of the object tree, but again, the search overhead would be too high.

Our solution is to maintain a more general mapping from input events to objects. Each context contains a list of input templates, each of which has an associated handling object. Input templates consist of a window identifier, device type, and finger identifier. If all elements of the template match corresponding elements of an input event (the template may have "don't care" values) then the event is sent to the indicated handling object. If no template matches, then input is sent to a default handling object, also specified in the current context. As a result, we can have:

- two fingers operating a knob (input from either finger is forwarded immediately to the knob object),

- another finger moving toward a switch (input from this finger goes to the root of the object tree as usual. The switch object may change the current context and take future input directly when the finger gets close), and

- a simultaneous mouse click on a button (this input would work its way through the object tree from the root to the button object).

In some cases, one might want to effect a global context change, such as a pop-up dialog box which preempts all controls. This is accomplished by pushing a new context on the stack. This may redirect input from an object with a gesture in progress. We avoid problems here by sending a "finger up" event to the old handling object and a "finger down" event to the new handling object whenever a finger changes windows.

4. Gesture Representation and Processing
Since individual finger coordinates do not convey any dynamic aspects of gestures, the first stage of processing Sensor Frame input data is to represent the path of each finger by a set of features. The features are then interpreted by controls. The current set of features includes a piece-wise linear approximation of the path, the point where the path first crosses into an "activation radius", and the cumulative angular change.

4.1. Initial Processing
The x,y,z coordinates are supplied by the Sensor Frame as integers but are translated to floating point for further processing. The x,y,z portion of the input data is referred to hereafter as a *Raw Data Point* or *RDP*.

Normally, the default handling object for RDP's is the root of the object tree. The tree is searched after each input; however, when the RDP falls within the bounding box of a control object, the object responds by putting a template in the current context that will direct future events with the same finger identifier to the object. Future matching events will arrive at the object where they are added to a table associated with both the object and the finger identifier. This table of RDP's is called an *open vector*.

4.2. Path Decomposition
The next step is to process the open vector of RDP's to obtain a segmented[1] representation. This representation simultaneously provides data reduction and immunity from jitter.

For convenience, we want our approximation to be continuous; that is, each segment begins where the previous one ended, and all endpoints coincide with data points (RDP's). The algorithm for constructing the approximation is straightforward: as each RDP is added to the open vector, and error measure is computed. When the error measure exceeds a constant threshold, a segment from the first to the next-to-last point is added to the path and the open vector is adjusted to contain the last two RDP's. This algorithm can be described as "greedy without backtracking" since we pack as many RDP's into each segment as possible (limited by the error threshold) and we

---

[1] In this discussion, a *segment* is an ordered pair of points, e.g. RDP's, and a point is an x, y, z triple.

never try alternative assignments of RDP's to segments.

Figure 4-1 illustrates the process. The segment from point 1 to point 3 falls below the error threshold, but a segment from point 1 to point 4 exceeds the threshold. Therefore, the segment [point 1, point 3] is added to the path, and a new open vector [point 3, point 4] is started. This is extended to point 5 and then to point 6.
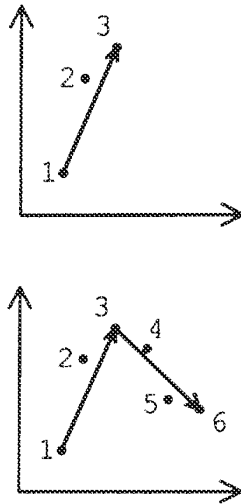

Figure 4-1: Fitting vectors to a set of points.

The error measure is:

$$error = \frac{1}{n}\sqrt{(\sum_{i=1}^{n} |D_x(p_i)|)^2 + (\sum_{i=1}^{n} |D_y(p_i)|)^2 + (\sum_{i=1}^{n} |D_z(p_i)|)^2}$$

where $D_x(p_i)$ is the x-component of the shortest vector from an RDP p_i to the proposed segment $[p_1, p_n]$ from point p_1 to p_n. We elected not to take a sum-of-squares in the innermost summation to save a bit of computation, and the resulting path decomposition seems to work well. The distance from a point to a line can be computed without trigonometric or square root functions as shown in Appendix I.

4.3. The Activation Volume
Gesture analysis is performed if an open vector passes into the volume defined by an activation radius and an activation center. Such processing will continue so long as succeeding RDP's remain within that volume.

An activation center is not necessarily static. For example, the knob on a slider has an activation center that moves along with it. The value associated with the device class is in this case a default initial value for the slider location.

Because we are polling the Sensor Frame from the application program, we cannot guarantee that we will catch all (or any) relevant RDP's within a possibly small *activation volume*. This is particularly true if the finger is traveling

quickly. However, by setting the size of the bounding box large enough, we can guarantee we will at least pick up endpoints of a path segment that intersects this volume. The same distance algorithm (see Appendix I) used for path decomposition is then used to see if the point of closest approach of the path to the *activation center* is less than the *activation radius*.

4.4. Gestures
Once an RDP falls within the activation radius, the gesture features are examined by the corresponding object. Response to gestures is programmed procedurally for each type of control.

A toggle switch (or any other control affected by a simple linear motion), can be moved if the direction of travel of a finger path (A) matches the preferred axis of travel of the device (B). We define a maximum angle (θ_{max}) between the two and see if the actual angle (θ_{act}) is within bounds.

The actual angular error can be found using the definition of the vector dot product:

$$A \cdot B = |A||B|\cos(\theta_{act})$$

and rearranging to solve for $\cos(\theta_{act})$:

$$\cos(\theta_{act}) = (A \cdot B)/(|A||B|)$$

If the inequality:

$$\cos(\theta_{act}) \le \cos(\theta_{max})$$

holds, then the movement of the finger is close enough to the preferred direction to cause a state change. Note that $\cos(\theta_{max})$ is a constant that can be precalculated, thus we avoid calculating transcendentals at run time by comparing cosines of angles instead of the angles themselves and by using the equation:

$$A \cdot B = A_x B_x + A_y B_y + A_z B_z$$

The knob rotation gesture consists of one or two fingers moving within the activation radius of the knob. Once it is determined that a finger path crosses the activation radius, an angle from the center of the knob to the finger is computed and saved. Each location change within the activation radius results in a recalculation of the angle, and the angle of the knob is updated by the angular difference. When there are two fingers within the activation radius, the knob is updated when either finger moves; the overall knob rotation is effectively the average rotation of the two fingers.

5. System Considerations

5.1. Implementation Languages
Our Sensor Frame interface, gesture recognition software, and graphics primitives are all implemented in the C programming language. Graphical and interactive objects, as well as the top-level input handling routines, are im-

130

plemented in XLISP, a lisp interpreter with built-in support for objects.

Although we would have preferred a compiled lisp, this work was begun at a time when our workstation environment was in a state of rapid change. During the course of the project, we ported XLISP to three machine types and implemented our graphics interface on two window managers. The fact that XLISP is a relatively small C program made it easy to port and to extend with the additional graphics and I/O primitives we needed.

5.2. Input Diagnostics
For diagnostic purposes, input of raw position data points is done through a device-independent module that allows input to come from a Sensor Frame, to be partially simulated by a mouse, or to be played back from a file that was "recorded" on a previous run with a mouse or a Sensor Frame. Bugs that appear only in long runs can be reproduced by playing back the log file during a debugging session.

The interface is implemented in such a way that regardless of which device is being used as the pointing device, the window menu is still available via the mouse. Commands are available to display every RDP as a small box on the screen; to print the results of every Sensor Frame input to a diagnostic window; to select a prerecorded file, a mouse or the Sensor Frame as the source of input; or to begin or end recording data for future playback.

6. Results and Conclusions
In the process of building GID, we have encountered several problems which are worth further study. One problem is how to organize prototyping software such as GID to allow controls to be operated in "run" mode and edited in "edit" mode. It seems inappropriate to implement editing within each object (Should a slider contain code for editing its size, placement, label, etc?), but a modular approach is preferable to a monolithic editor that captures all input in edit mode. In GID, we divert input when in "edit" mode, but we have specific editing methods in various subclasses of Control. One alternative is to implement all interactive behavior outside of control objects as in Garnet [8].

Another problem is that we have no high-level procedures for recognizing complex gestures: our recognizers must be hand-coded using fairly low-level representations. A promising alternative is the pattern recognition approach being pursued by Dean Rubine [10].

We know of no window managers that support multiple cursors. Ideally, the window manager should track each finger with a cursor and also determine what window contains each visible finger. Currently, the overhead of cursor tracking and mapping input to windows from outside of the window manager (X11) causes significant performance problems.

The present resolution of the Sensor Frame is only about 160 x 200 points. While this provides plenty of resolution relative to the size of controls displayed on the screen, greater resolution is needed in order to accurately measure the direction of motion and to minimize jitter.

The organization of GID prevents a single gesture from being received by multiple controls simultaneously. We do not feel this is a serious limitation, but it could be avoided by utilizing a more complete mapping from RDP's to objects. Rather than searching the object tree depth first, we could use hashing or a linear search of all objects to locate potentially overlapping bounding boxes which contain each RDP. Input events would then be duplicated and sent to each "interested" object. This technique was tried in an earlier system and allowed, for example, two adjacent switches to be flipped by moving a finger between them.

We note that some window managers might assist in the implementation of controls: if each control is implemented as a sub-window, then the search for a handler could be performed by the window manager. This technique will *not* work if we want input to reach multiple controls because current window managers will map input to only one window even if there is overlap. Furthermore, window managers typically assume a single pointing device, and extensive modification would be required to handle input from the Sensor Frame or some other gesture sensing device.

In conclusion, we have implemented a system for prototyping gesture-based user interfaces. The system is capable of editing its own interface, and applications are typically built by extension. The system allows us to experiment with screen layout and with multiple input devices without programming, and the system is extensible so that new interaction techniques can be integrated and evaluated. We have found piecewise linear approximations to paths to be an appropriate representation for simple gestures, and our vector software can be reused by different control objects.

7. Acknowledgments

References

1. R. A. Bolt. *The Human Interface: where people and computers meet*. Lifetime Learning Publications, 1984.

2. Frederick P. Brooks, Jr. Grasping Reality Through Illusion - Interactive Graphics Serving Science. CHI '88 Proceedings, May, 1988, pp. 1-11.

3. Luca Cardelli. Building User Interfaces by Direct Manipulation. Tech. Rept. 22, Digital Equipment Corporation Systems Research Center Research Report, Oct., 1987.

4. S. S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual Environment Display System. ACM Workshop on Interactive 3D Grahpics, Association for Computing Machinery, 1986, pp. 77-87.

5. Jean-Marie Hullot. *Interface Builder*. Santa Barbara, CA, 1987.

6. Myron W. Krueger. *Artificial Reality*. Addison-Wesley, Reading, MA, 1983.

7. Paul McAvinney. U.S. Patent No. 4,746,770; Method and Apparatus for Isolating and Manipulating Graphic Objects On Computer Video Monitor. May 24, 1988.

8. Brad A. Myers. Encapsulating Interactive Behaviors. Human Factors in Computing Systems, SIGCHI'89, Austin, TX, April, 1989, pp. (to appear).

9. NeXT, Inc. *Interface Builder*. Palo Alto, CA, 1988. (online, preliminary documentation).

10. Dean Rubine. The Automatic Recognition of Gestures. (thesis proposal, Carnegie Mellon University School of Computer Science).

11. Kurt J. Schmucker. *Object-oriented programming for the Macintosh*. Hayden Book Co., Hasbrouck Heights, N.J., 1986.

12. David Weimer and S. K. Ganapathy. A Synthetic Visual Environment With Hand Gesturing and Voice Input. CHI'89 Conference Proceedings, Association for Computing Machinery's Special Interest Group on Computer Human Interaction, 1989, pp. 235-240.

I. Minimum Distance Between a Point and Segment

```
                X
                |
                |
                |
A---------------P-------------------------B
```
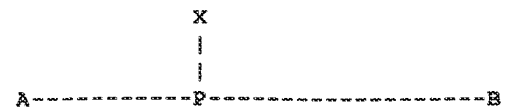
Parameterize equation of AB:

$$V(k) = (1-k)A + kB$$

for $0 \leq k \leq 1$, and $A \leq V \leq B$ so that V is any point on the segment between A and B.

Release the constraint on k for the time being, and let P be the point nearest X on AB: $P = V(k_p)$.

This gives us Equation 1:

$$Eqn\ 1 \quad P = (1-k_p)A + k_p B$$

or, in expanded form:

$$P = A - k_p A + k_p B$$

We want a line normal to AB that passes through X. By definition the dot product is zero if $\angle APX = 90°$, so for $XP \perp AP$ we have:

$$(P-X) \cdot (P-A) = 0$$

Now substitute for P:

$$(A - k_p A + k_p B - X) \cdot (-k_p A + k_p B) = 0$$

expand terms:

$$(-k_p + k_p^2)(A \cdot A) + (-k_p^2 + k_p - k_p^2)(A \cdot B) +$$
$$(k_p^2)(B \cdot B) + k_p(A \cdot X) - k_p(B \cdot X) = 0$$

divide through by k_p and simplify:

$$(-1 + k_p)(A \cdot A) + (-2k_p + 1)(A \cdot B) +$$
$$k_p(B \cdot B) + (A \cdot X) - (B \cdot X) = 0$$

arrange terms for easier reduction:

$$-(1 - k_p)(A \cdot A) + [(-k_p + 1)(A \cdot B) - k_p(A \cdot B)] +$$
$$k_p(B \cdot B) + (A \cdot X) - (B \cdot X) = 0$$

apply distributive property of dot product:

$$(1 - k_p)[A \cdot (B - A)] + k_p[(B - A) \cdot B] = [X \cdot (B - A)]$$

collect terms:

$$k_p[[-A \cdot (B - A)] + [(B - A) \cdot B]] + [A \cdot (B - A)] = [X \cdot (B - A)]$$

apply distributive property of dot product again:

$$k_p[(B - A) \cdot (B - A)] = [(X - A) \cdot (B - A)]$$

solve for k_p:

$$Eqn\ 2 \quad k_p = \frac{(B - A) \cdot (X - A)}{(B - A) \cdot (B - A)}$$

Note that if $k_p < 0$, the nearest point to X is A. If $k_p > 1$, it is B. Otherwise solve Eqn 1 with value of k_p from Eqn 2 to get the nearest point.

Page 1674 of 1714

Appendix B: Sensor Frame UNIX Device Driver Library Functions

Following is a list of the Sensor Frame UNIX device driver C-callable functions:

```
sf_open( connection )
sf_close( sf_fd )

sf_perror( string )

sf_scale( sf_fd, xmin, xmax, ymin, ymax, zmin, zmax )
sf_query_scale( sf_fd, xmin, xmax, ymin, ymax, zmin, zmax )

sf_enable( sf_fd, types, boolean )
sf_q_enable( sf_fd, types )

sf_queue( sf_fd, types, boolean )
sf_q_queue( sf_fd, types )

sf_poll_once( event_structure )
sf_poll_all( boolean, sf_fd, event_structure )

sf_qtest( )
sf_qread( event_structure )
sf_qflush( )
sf_qadd( event_structure )
sf_qpush( event_structure )

sf_user_input_handler( sf_fd, user_function_address )

sf_filter( sf_fd, filter_structure )
sf_q_filter( sf_fd, filter_id, filter_structure )

sf_toss( )
```

Appendix C: Contents of Sensor Frame Videotape (VHS Format)

Appendix C consists of a VHS Videotape showing the Following:
- Appendix C-1: The Sensor Frame
- Appendix C-2: The First and Second Prototype Sensor Cubes
- Appendix C-3: The Gesture Based User Interface Prototyping System (GID)
- Appendix C-4: The VideoHarp

Copies of the Videotape are available upon request from Sensor Frame Corporation.

ThinSight: A Thin Form-Factor Interactive Surface Technology

By Shahram Izadi, Steve Hodges, Alex Butler, Darren West, Alban Rrustemi, Mike Molloy and William Buxton

ABSTRACT

ThinSight is a thin form-factor interactive surface technology based on optical sensors embedded inside a regular liquid crystal display (LCD). These augment the display with the ability to sense a variety of objects near the surface, including fingertips and hands, to enable multitouch interaction. Optical sensing also allows other physical items to be detected, allowing interactions using various tangible objects. A major advantage of ThinSight over existing camera and projector-based systems is its compact form-factor, making it easier to deploy in a variety of settings. We describe how the ThinSight hardware is embedded behind a regular LCD, allowing sensing without degradation of display capability, and illustrate the capabilities of our system through a number of proof-of-concept hardware prototypes and applications.

1. INTRODUCTION

Touch input using a single point of contact with a display is a natural and established technique for human computer interaction. Research over the past decades,[3] and more recently products such as the iPhone and Microsoft Surface, have shown the novel and exciting interaction techniques and applications possible if multiple simultaneous touch points can be detected.

Various technologies have been proposed for multitouch sensing in this way, some of which extend to detection of physical objects in addition to fingertips. Systems based on optical sensing have proven to be particularly powerful in the richness of data captured and the flexibility they can provide. As yet, however, such optical systems have predominately been based on cameras and projectors and require a large optical path in front of or behind the display. This typically results in relatively bulky systems—something that can impact adoption in many real-world scenarios. While capacitive overlay technologies, such as those in the iPhone and the Dell XT Tablet PC, can support thin form-factor multitouch, they are limited to sensing only fingertips.

ThinSight is a novel interactive surface technology which is based on optical sensors integrated into a thin form-factor LCD. It is capable of imaging multiple fingertips, whole hands, and other objects near the display surface as shown in Figure 1. The system is based upon custom hardware embedded behind an LCD, and uses infrared (IR) light for sensing without degradation of display capability.

In this article we describe the ThinSight electronics and the modified LCD construction which results. We present two prototype systems we have developed: a multitouch laptop and a touch-and-tangible tabletop (both shown in Figure 1). These

systems generate rich sensor data which can be processed using established computer vision techniques to prototype a wide range of interactive surface applications.

As shown in Figure 1, the shapes of many physical objects, including fingers, brushes, dials, and so forth, can be "seen" when they are near the display, allowing them to enhance multitouch interactions. Furthermore, ThinSight allows interactions close-up or at a distance using active IR pointing devices, such as styluses, and enables IR-based communication through the display with other electronic devices.

We believe that ThinSight provides a glimpse of a future where display technologies such as LCDs and organic light emitting diodes (OLEDs) will cheaply incorporate optical sensing pixels alongside red, green and blue (RGB) pixels in

Figure 1. ThinSight brings the novel capabilities of surface computing to thin displays. Top left: photo manipulation using multiple fingers on a laptop prototype (note the screen has been reversed in the style of a Tablet PC). Top right: a hand, mobile phone, remote control and reel of tape placed on a tabletop ThinSight prototype, with corresponding sensor data far right. Note how all the objects are imaged through the display, potentially allowing not only multitouch but tangible input. Bottom left and right: an example of how such sensing can be used to support digital painting using multiple fingertips, a real brush and a tangible palette to change paint colors.
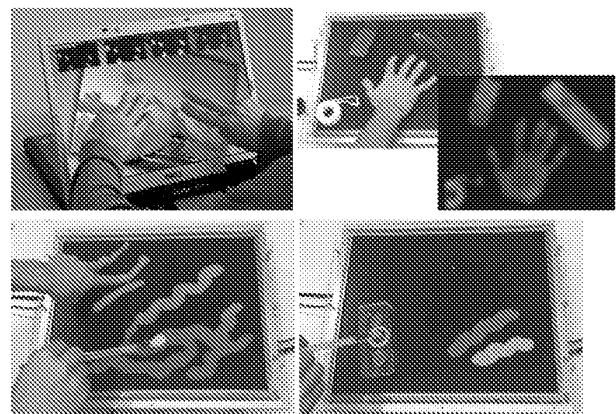
Page 1677 of 1714

a similar manner, resulting in the widespread adoption of such surface technologies.

2. OVERVIEW OF OPERATION

2.1. Imaging through an LCD using IR light
A key element in the construction of ThinSight is a device known as a retro-reflective optosensor. This is a sensing element which contains two components: a light emitter and an optically isolated light detector. It is therefore capable of both emitting light and, at the same time, detecting the intensity of incident light. If a reflective object is placed in front of the optosensor, some of the emitted light will be reflected back and will therefore be detected.

ThinSight is based around a 2D grid of retro-reflective optosensors which are placed behind an LCD panel. Each optosensor emits light that passes right through the entire panel. Any reflective object in front of the display (such as a fingertip) will reflect a fraction of the light back, and this can be detected. Figure 2 depicts this arrangement. By using a suitably spaced grid of retro-reflective optosensors distributed uniformly behind the display it is therefore possible to detect any number of fingertips on the display surface. The raw data generated is essentially a low resolution grayscale "image" of what can be seen through the display, which can be processed using computer vision techniques to support touch and other input.

A critical aspect of ThinSight is the use of retro-reflective sensors that operate in the infrared part of the spectrum, for three main reasons:
- Although IR light is attenuated by the layers in the LCD panel, some still passes through the display.[5] This is largely unaffected by the displayed image.
- A human fingertip typically reflects around 20% of incident IR light and is therefore a quite passable "reflective object."
- IR light is not visible to the user, and therefore does not detract from the image being displayed on the panel.

Figure 2. The basic principle of ThinSight. An array of retro-reflective optosensors is placed behind an LCD. Each of these contains two elements: an emitter which shines IR light through the panel; and a detector which picks up any light reflected by objects such as fingertips in front of the screen.
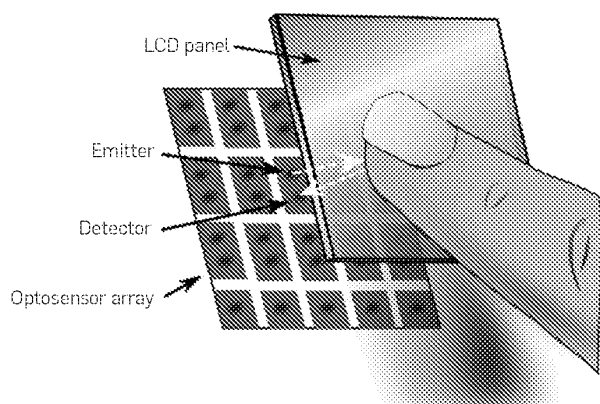
2.2. Further features of ThinSight
ThinSight is not limited to detecting fingertips in contact with the display; any suitably reflective object will cause IR light to reflect back and will therefore generate a "silhouette." Not only can this be used to determine the location of the object on the display, but also its orientation and shape, within the limits of sensing resolution. Furthermore, the underside of an object may be augmented with a visual mark—a barcode of sorts—to aid identification.

In addition to the detection of passive objects via their shape or some kind of barcode, it is also possible to embed a very small infrared transmitter into an object. In this way, the object can transmit a code representing its identity, its state, or some other information, and this data transmission can be picked up by the IR detectors built into ThinSight. Indeed, ThinSight naturally supports bidirectional IR-based data transfer with nearby electronic devices such as smartphones and PDAs. Data can be transmitted from the display to a device by modulating the IR light emitted. With a large display, it is possible to support several simultaneous bidirectional communication channels in a spatially multiplexed fashion.

Finally, a device which emits a collimated beam of IR light may be used as a pointing device, either close to the display surface like a stylus, or from some distance. Such a pointing device could be used to support gestures for new forms of interaction with a single display or with multiple displays. Multiple pointing devices could be differentiated by modulating the light generated by each.

3. THE THINSIGHT HARDWARE

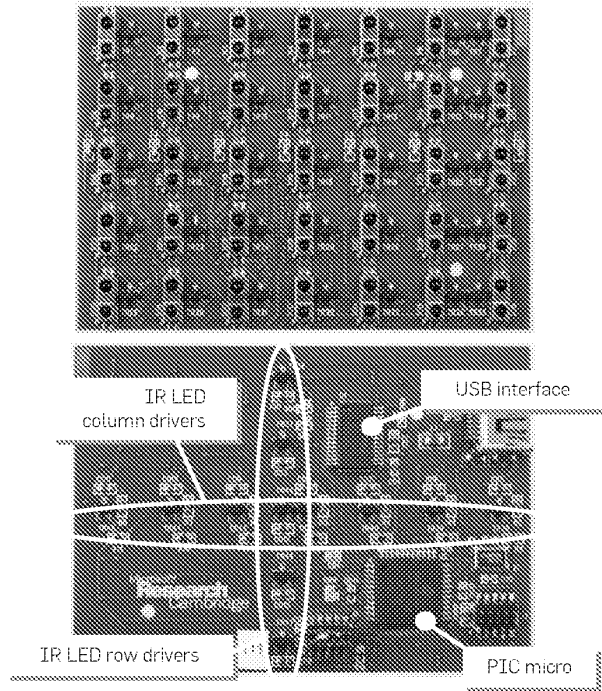### 3.1. The sensing electronics
The prototype ThinSight circuit board depicted in Figure 3 uses Avago HSDL-9100 retro-reflective infrared sensors. These devices are especially designed for proximity sensing —an IR LED emits infrared light and an IR photodiode generates a photocurrent which varies with the amount of incident light. Both emitter and detector have a center wavelength of 940 nm.

A 7×5 grid of these HSDL-9100 devices on a regular 10 mm pitch is mounted on custom-made 70×50 mm 4-layer printed circuit board (PCB). Multiple PCBs can be tiled together to support larger sensing areas. The IR detectors are interfaced directly with digital input/output lines on a PIC18LF4520 microcontroller.

The PIC firmware collects data from one row of detectors at a time to construct a "frame" of data which is then transmitted to the PC over USB via a virtual COM port. To connect multiple PCBs to the same PC, they must be synchronized to ensure that IR emitted by a row of devices on one PCB does not adversely affect scanning on a neighboring PCB. In our prototype we achieve this using frame and row synchronization signals which are generated by one of the PCBs (the designated "master") and detected by the others ("slaves").

Note that more information on the hardware can be found in the full research publications.[7,10]
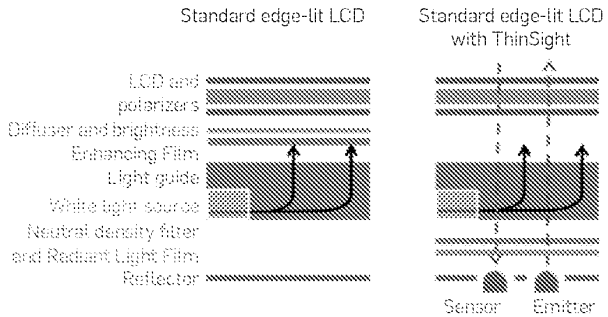
Page 1678 of 1714

Figure 3. Top: the front side of the sensor PCB showing the 7×5 array of IR optosensors. The transistors that enable each detector are visible to the right of each optosensor. Bottom: the back of the sensor PCB has little more than a PIC microcontroller, a USB interface and FETs to drive the rows and columns of IR emitting LEDs. Three such PCBs are used in our ThinSight laptop while there are thirty in the tabletop prototype.

Figure 4. Typical LCD edge-lit architecture shown left. The LCD comprises a stack of optical elements. A white light source is typically located along one or two edges at the back of the panel. A white reflector and transparent light guide direct the light toward the front of the panel. The films help scatter this light uniformly and enhance brightness. However, they also cause excessive attenuation of IR light. In ThinSight, shown right, the films are substituted and placed behind the light guide to minimize attenuation and also reduce noise caused by LCD flexing upon touch. The sensors and emitters are placed at the bottom of the resulting stack, aligned with holes cut in the reflector.

3.2. LCD technology overview

To understand how the ThinSight hardware is integrated into a display panel, it is useful to understand the construction and operation of a typical LCD. An LCD panel is made up of a stack of optical components as shown in Figure 4. At the front of the panel is a thin layer of liquid crystal material which is sandwiched between two polarizers. The polarizers are orthogonal to each other, which means that any light which passes through the first will naturally be blocked by the second, resulting in dark pixels. However, if a voltage is applied across the liquid crystal material at a certain pixel location, the polarization of light incident on that pixel is twisted through 90° as it passes through the crystal structure. As a result it emerges from the crystal with the correct polarization to pass through the second polarizer. Typically, white light is shone through the panel from behind by a backlight and red, green, and blue filters are used to create a color display. In order to achieve a low profile construction while maintaining uniform lighting across the entire display and keeping cost down, the backlight is often a large "light guide" in the form of a clear acrylic sheet which sits behind the entire LCD and which is edge-lit from one or more sides. The light source is often a cold cathode fluorescent tube or an array of white LEDs. To maximize the efficiency and uniformity of the lighting, additional layers of material may

be placed between the light guide and the LCD. Brightness enhancing film (BEF) "recycles" visible light at suboptimal angles and polarizations and a diffuser smoothes out any local nonuniformities in light intensity.

3.3. Integration with an LCD panel

We constructed our ThinSight prototypes using a variety of desktop and laptop LCD panels, ranging from 17" to 21". Two of these are shown in Figures 5 and 6. Up to 30 PCBs were tiled to support sensing across the entire surface. In instances where large numbers of PCBs were tiled, a custom hub circuit based on an FPGA was designed to collect and aggregate the raw data captured from a number of tiled sensors and transfer this to the PC using a single USB channel. These tiled PCBs are mounted directly behind the light guide. To ensure that the cold cathode does not cause any stray IR light to emanate from the acrylic light guide, we placed a narrow piece of IR-blocking film between it and the backlight. We cut small holes in the white reflector behind the light guide to coincide with the location of every IR emitting and detecting element.

During our experiments we found that the combination of the diffuser and BEF in an LCD panel typically caused excessive attenuation of the IR signal. However, removing these materials degrades the displayed image significantly: without BEF the brightness and contrast of the displayed image is reduced unacceptably; without a diffuser the image appears to "float" in front of the backlight and at the same time the position of the IR emitters and detectors can be seen in the form of an array of faint dots across the entire display.

To completely hide the IR emitters and detectors we required a material that lets IR pass through it but not visible light, so that the optosensors could not be seen but would operate normally. The traditional solution would be

Figure 5. Our laptop prototype. Top: Three PCBs are tiled together and mounted on an acrylic plate, to give a total of 105 sensing pixels. Holes are also cut in the white reflector shown on the far left. Bottom left: an aperture is cut in the laptop lid to allow the PCBs to be mounted behind the LCD. This provides sensing across the center of the laptop screen. Bottom right: side views of the prototype—note the display has been reversed on its hinges in the style of a Tablet PC.
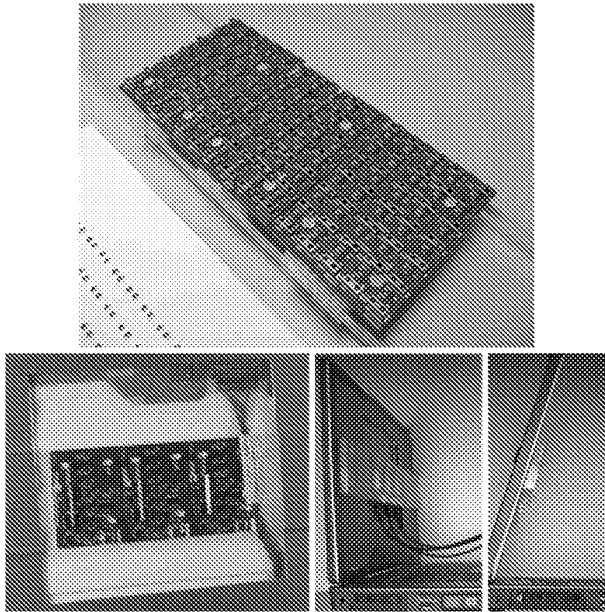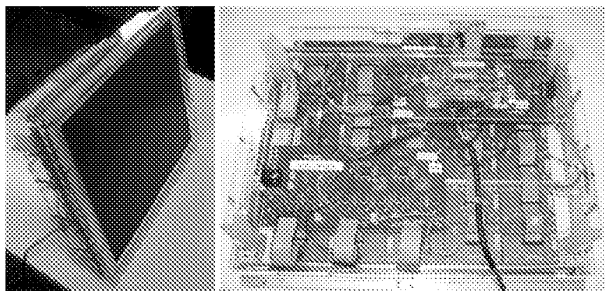

Figure 6. The ThinSight tabletop hardware as viewed from the side and behind. Thirty PCBs (in a 5×6 grid) are tiled with columns interconnected with ribbon cable and attached to a hub board for aggregating data and inter-tile communication. This provides a total of 1050 discrete sensing pixels across the entire surface.

to use what is referred to as a "cold mirror." Unfortunately these are made using a glass substrate which means they are expensive, rigid and fragile and we were unable to source a cold mirror large enough to cover the entire tabletop display. We experimented with many alternative materials including tracing paper, acetate sheets coated in emulsion paint, spray-on frosting, thin sheets of white polythene and mylar. Most of these are unsuitable either because of

a lack of IR transparency or because the optosensors can be seen through them to some extent. The solution we settled on was the use of Radiant Light Film by 3M (part number CM500), which largely lets IR light pass through while reflecting visible light without the disadvantages of a true cold mirror. This was combined with the use of a grade "0" neutral density filter, a visually opaque but IR transparent diffuser, to even out the distribution rear illumination and at the same time prevent the "floating" effect. Applying the Radiant Light Film carefully is critical since minor imperfections (e.g. wrinkles or bubbles) are highly visible to the user—thus we laminated it onto a thin PET carrier. One final modification to the LCD construction was to deploy these films *behind* the light guide to further improve the optical properties. The resulting LCD layer stack-up is depicted in Figure 4 right.

Most LCD panels are not constructed to resist physical pressure, and any distortion which results from touch interactions typically causes internal IR reflection resulting in "flare." Placing the Radiant Light Film and neutral density filter behind the light guide improves this situation, and we also reinforced the ThinSight unit using several lengths of extruded aluminum section running directly behind the LCD.

4. THINSIGHT IN OPERATION

4.1. Processing the raw sensor data
Each value read from an individual IR detector is defined as an integer representing the intensity of incident light. These sensor values are streamed to the PC via USB where the raw data undergoes several simple processing and filtering steps in order to generate an IR image that can be used to detect objects near the surface. Once this image is generated, established image processing techniques can be applied in order to determine coordinates of fingers, recognize hand gestures, and identify object shapes.

Variations between optosensors due to manufacturing and assembly tolerances result in a range of different values across the display even without the presence of objects on the display surface. To make the sensor image uniform and the presence of additional incident light (reflected from nearby objects) more apparent, we subtract a "background" frame captured when no objects are present, and normalize relative to the image generated when the display is covered with a sheet of white reflective paper.

We use standard bicubic interpolation to scale up the sensor image by a predefined factor (10 in our current implementation). For the larger tabletop implementation this results in a 350 × 300 pixel image. Optionally, a Gaussian filter can be applied for further smoothing, resulting in a grayscale "depth" image as shown in Figure 7.

4.2. Seeing through the ThinSight display
The images we obtain from the prototype are quite rich, particularly given the density of the sensor array. Fingers and hands within proximity of the screen are clearly identifiable. Examples of images captured through the display are shown in Figures 1, 7 and 8.

Page 1680 of 1714

Figure 7. The raw ThinSight sensor data shown left and after interpolation and smoothing right. Note that the raw image is a very low resolution, but contains enough data to generate the relatively rich image at right.
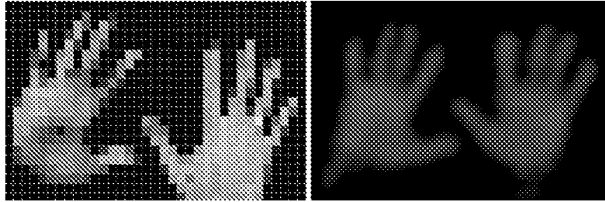

Figure 8. Fingertips can be sensed easily with ThinSight. Left: the user places five fingers on the display to manipulate a photo. Right: a close-up of the sensor data when fingers are positioned as shown at left. The raw sensor data is: (1) scaled-up with interpolation, (2) normalized, (3) thresholded to produce a binary image, and finally (4) processed using connected components analysis to reveal the fingertip locations.
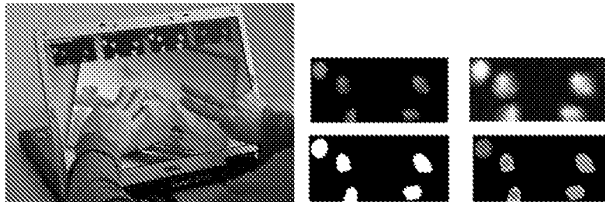
Fingertips appear as small blobs in the image as they approach the surface, increasing in intensity as they get closer. This gives rise to the possibility of sensing both touch and hover. To date we have only implemented touch/no-touch differentiation, using thresholding. However, we can reliably and consistently detect touch to within a few millimeters for a variety of skin tones, so we believe that disambiguating hover from touch would be possible.

In addition to fingers and hands, optical sensing allows us to observe other IR reflective objects through the display. Figure 1 illustrates how the display can distinguish the shape of many reflective objects in front of the surface, including an entire hand, mobile phone, remote control, and a reel of white tape. We have found in practice that many objects reflect IR.

A logical next step is to attempt to uniquely identify objects by placement of visual codes underneath them. Such codes have been used effectively in tabletop systems such as the Microsoft Surface and various research prototypes[12, 28] to support tangible interaction. We have also started preliminary experiments with the use of such codes on ThinSight, see Figure 9.

Active electronic identification schemes are also feasible. For example, cheap and small dedicated electronic units containing an IR emitter can be stuck onto or embedded inside objects that need to be identified. These emitters will produce a signal directed to a small subset of the display sensors. By emitting modulated IR it is possible to transmit a unique identifier to the display.

4.3. Communicating through the ThinSight display

Beyond simple identification, an embedded IR transmitter also provides a basis for supporting richer bidirectional communication with the display. In theory any IR modulation scheme, such as the widely adopted IrDA standard, could be supported by ThinSight. We have implemented a DC-balanced modulation scheme which allows retroreflective object sensing to occur *at the same time* as data transmission. This required no additions or alterations to the sensor PCB, only changes to the microcontroller firmware. To demonstrate our prototype implementation of this, we built a small embedded IR transceiver based on a low power MSP430 microcontroller, see Figure 10. We encode 3 bits of data in the IR transmitted from the ThinSight pixels to control an RGB LED fitted to the embedded receiver. When the user touches various soft buttons on the ThinSight display, this in turn transmits different 3 bit codes from ThinSight pixels to cause different colors on the embedded device to be activated.

It is theoretically possible to transmit and receive different data simultaneously using different columns on the

Figure 9. An example 2" diameter visual marker and the resulting ThinSight image after processing.
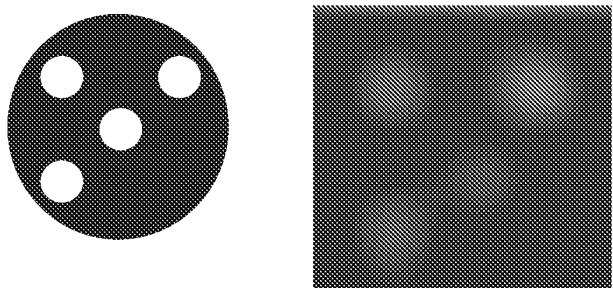

Figure 10. Using ThinSight to communicate with devices using IR. Top left: an embedded microcontroller/IR transceiver/RGB LED device. Bottom left: touching a soft button on the ThinSight display signals the RGB LED on the embedded device to turn red (bottom right). Top right: A remote control is used to signal from a distance the display whichin turn sends an IR command to the RGB device to turn the LED blue.
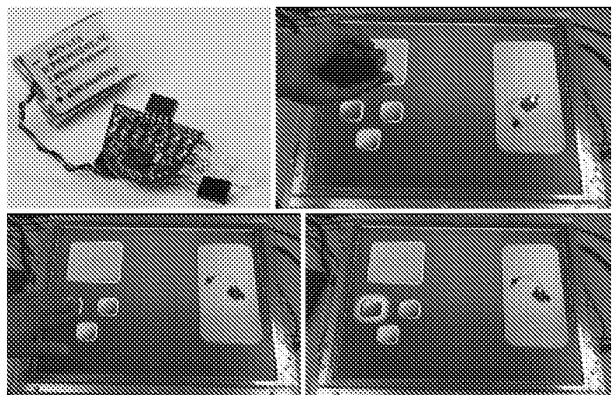
display surface, thereby supporting spatially multiplexed bidirectional communications with multiple local devices and reception of data from remote gesturing devices. Of course, it is also possible to time multiplex communications between different devices if a suitable addressing scheme is used. We have not yet prototyped either of these multiple-device communications schemes.

4.4. Interacting with ThinSight
As shown earlier in this section, it is straightforward to sense and locate multiple fingertips using ThinSight. In order to do this we threshold the processed data to produce a binary image. The connected components within this are isolated, and the center of mass of each component is calculated to generate representative X, Y coordinates of each finger. A very simple homography can then be applied to map these fingertip positions (which are relative to the sensor image) to onscreen coordinates. Major and minor axis analysis or more detailed shape analysis can be performed to determine orientation information. Robust fingertip tracking algorithms or optical flow techniques[28] can be employed to add stronger heuristics for recognizing gestures.

Using these established techniques, fingertips are sensed to within a few millimeters, currently at 23 frames/s. Both hover and touch can be detected, and could be disambiguated by defining appropriate thresholds. A user therefore need not apply any force to interact with the display. However, it is also possible to estimate fingertip pressure by calculating the increase in the area and intensity of the fingertip "blob" once touch has been detected.

Figure 1 shows two simple applications developed using ThinSight. A simple photo application allows multiple images to be translated, rotated, and scaled using established multifinger manipulation gestures. We use distance and angle between touch points to compute scale factor and rotation deltas. To demonstrate some of the capabilities of ThinSight beyond just multitouch, we have built an example paint application that allows users to paint directly on the surface using both fingertips and real paint brushes. The latter works because ThinSight can detect the brushes' white bristles which reflect IR. The paint application also supports a more sophisticated scenario where an artist's palette is placed on the display surface. Although this is visibly transparent, it has an IR reflective marker on the underside which allows it to be detected by ThinSight, whereupon a range of paint colors are rendered underneath it. The user can change color by "dipping" either a fingertip or a brush into the appropriate well in the palette. We identify the presence of this object using a simple ellipse matching algorithm which distinguishes the larger palette from smaller touch point "blobs" in the sensor image. Despite the limited resolution of ThinSight, it is possible to differentiate a number of different objects using simple silhouette shape information.

5. DISCUSSION AND FUTURE WORK
We believe that the prototype presented in this article is an interesting proof-of-concept of a new approach to multi-touch and tangible sensing for thin displays. We have already described some of its potential; here we discuss a number of additional observations and ideas which came to light during the work.

5.1. Fidelity of sensing
The original aim of this project was simply to detect fingertips to enable multi-touch-based direct manipulation. However, despite the low resolution of the raw sensor data, we still detect quite sophisticated object images. Very small objects do currently "disappear" on occasion when they are midway between optosensors. However, we have a number of ideas for improving the fidelity further, both to support smaller objects and to make object and visual marker identification more practical. An obvious solution is to increase the density of the optosensors, or at least the density of IR detectors. Another idea is to measure the amount of reflected light under different lighting conditions—for example, simultaneously emitting light from neighboring sensors is likely to cause enough reflection to detect smaller objects.

5.2. Frame rate
In informal trials of ThinSight for a direct manipulation task, we found that the current frame rate was reasonably acceptable to users. However, a higher frame rate would not only produce a more responsive UI which will be important for some applications, but would make temporal filtering more practical thereby reducing noise and improving sub-pixel accuracy. It would also be possible to sample each detector under a number of different illumination conditions as described above, which we believe would increase fidelity of operation.

5.3. Robustness to lighting conditions
The retro-reflective nature of operation of ThinSight combined with the use of background substitution seems to give reliable operation in a variety of lighting conditions, including an office environment with some ambient sunlight. One common approach to mitigating any negative effects of ambient light, which we could explore if necessary, is to emit modulated IR and to ignore any nonmodulated offset in the detected signal.

5.4. Power consumption
The biggest contributor to power consumption in ThinSight is emission of IR light; because the signal is attenuated in both directions as it passes through the layers of the LCD panel, a high intensity emission is required. For mobile devices, where power consumption is an issue, we have ideas for improvements. We believe it is possible to enhance the IR transmission properties of an LCD panel by optimizing the materials used in its construction for this purpose—something which is not currently done. In addition, it may be possible to keep track of object and fingertip positions, and limit the most frequent IR emissions to those areas. The rest of the display would be scanned less frequently (e.g. at 2–3 frames/s) to detect new touch points.

One of the main ways we feel we can improve on power consumption and fidelity of sensing is to use a more

Page 1682 of 1714

sophisticated IR illumination scheme. We have been experimenting with using an acrylic overlay *on top* of the LCD and using IR LEDs for edge illumination. This would allow us to sense multiple touch points using standard Frustrated Total Internal Reflection (FTIR),[5] but not objects. We have, however, also experimented with a material called Endlighten which allows this FTIR scheme to be extended to diffuse illumination, allowing both multitouch and object sensing with far fewer IR emitters than our current setup. The overlay can also serve the dual purpose of protecting the LCD from flexing under touch.

6. RELATED WORK

The area of interactive surfaces has gained particular attention recently following the advent of the iPhone and Microsoft Surface. However, it is a field with over two decades of history.[3] Despite this sustained interest there has been an evident lack of off-the-shelf solutions for detecting multiple fingers and/or objects on a display surface. Here, we summarize the relevant research in these areas and describe the few commercially available systems.

6.1. Camera-based systems

One approach to detecting multitouch and tangible input is to use a video camera placed in front of or above the surface, and apply computer vision algorithms for sensing. Early seminal work includes Krueger's VideoDesk[13] and the DigitalDesk,[26] which use dwell time and a microphone (respectively) to detect when a user is actually touching the surface. More recently, the Visual Touchpad[17] and C-Slate[9] use a stereo camera placed above the display to more accurately detect touch. The disparity between the image pairs determines the height of fingers above the surface. PlayAnywhere[28] introduces a number of additional image processing techniques for front-projected vision-based systems, including a shadow-based touch detection algorithm, a novel visual bar code scheme, paper tracking, and an optical flow algorithm for bimanual interaction.

Camera-based systems such as those described above obviously require direct line-of-sight to the objects being sensed which in some cases can restrict usage scenarios. Occlusion problems are mitigated in PlayAnywhere by mounting the camera off-axis. A natural progression is to mount the camera *behind* the display. HoloWall[18] uses IR illuminant and a camera equipped with an IR pass filter behind a diffusive projection panel to detect hands and other IR-reflective objects in front of it. The system can accurately determine the contact areas by simply thresholding the infrared image. TouchLight[27] uses rear-projection onto a holographic screen, which is also illuminated from behind with IR light. A number of multitouch application scenarios are enabled including high-resolution imaging capabilities. Han[5] describes a straightforward yet powerful technique for enabling high-resolution multitouch sensing on rear-projected surfaces based on FTIR. Compelling multitouch applications have been demonstrated using this technique. The Smart Table[22] uses this same FTIR technique in a tabletop form factor.

The Microsoft Surface and ReacTable[12] also use rear-projection, IR illuminant and a rear mounted IR camera to monitor fingertips, this time in a horizontal tabletop form-factor. These systems also detect and identify objects with IR-reflective markers on their surface.

The rich data generated by camera-based systems provides extreme flexibility. However, as Wilson discusses[28] this flexibility comes at a cost, including the computational demands of processing high resolution images, susceptibility to adverse lighting conditions and problems of motion blur. However, perhaps more importantly, these systems require the camera to be placed at some distance from the display to capture the entire scene, limiting their portability, practicality and introducing a setup and calibration cost.

6.2. Opaque embedded sensing

Despite the power of camera-based systems, the associated drawbacks outlined above have resulted in a number of parallel research efforts to develop a non-vision-based multitouch display. One approach is to embed a multitouch sensor of some kind behind a surface that can have an image projected onto it. A natural technology for this is capacitive sensing, where the capacitive coupling to ground introduced by a fingertip is detected, typically by monitoring the rate of leakage of charge away from conductive plates or wires mounted behind the display surface.

Some manufacturers such as Logitech and Apple have enhanced the standard laptop-style touch pad to detect certain gestures based on more than one point of touch. However, in these systems, using more than two or three fingers typically results in ambiguities in the sensed data. This constrains the gestures they support. Lee et al.[14] used capacitive sensing with a number of discrete metal electrodes arranged in a matrix configuration to support multitouch over a larger area. Westerman[25] describes a sophisticated capacitive multitouch system which generates x-ray-like images of a hand interacting with an opaque sensing surface, which could be projected onto. A derivative of this work was commercialized by Fingerworks.

DiamondTouch[4] is composed of a grid of row and column antennas which emit signals that capacitively couple with users when they touch the surface. Users are also capacitively coupled to receivers through pads on their chairs. In this way the system can identify which antennas behind the display surface are being touched and by which user, although a user touching the surface at two points can produce ambiguities. The SmartSkin[21] system consists of a grid of capacitively coupled transmitting and receiving antennas. As a finger approaches an intersection point, this causes a drop in coupling which is measured to determine finger proximity. The system is capable of supporting multiple points of contact by the same user and generating images of contact regions of the hand. SmartSkin and DiamondTouch also support physical objects, but can only identify an object when a user touches it. Tactex provide another interesting example of an opaque multitouch sensor, which uses transducers to measure surface pressure at multiple touch points.[23]

Page 1683 of 1714

6.3. Transparent overlays

The systems above share one major disadvantage: they all rely on front-projection for display. The displayed image will therefore be broken up by the user's fingers, hands and arms, which can degrade the user experience. Also, a large throw distance is typically required for projection which limits portability. Furthermore, physical objects can only be detected in limited ways, if object detection is supported at all.

One alternative approach to address some of the issues of display and portability is to use a transparent sensing overlay in conjunction with a self-contained (i.e., not projected) display such as an LCD panel. DualTouch[19] uses an off-the-shelf transparent resistive touch overlay to detect the position of two fingers. Such overlays typically report the average position when two fingers are touching. Assuming that one finger makes contact first and does not subsequently move, the position of a second touch point can be calculated. An extension to this is provided by Loviscach.[16]

The Philips Entertaible[15] takes a different "overlay" approach to detect up to 30 touch points. IR emitters and detectors are placed on a bezel around the screen. Breaks in the IR beams detect fingers and objects. The SMART DViT[22] and HP TouchSmart[6] utilize cameras in the corners of a bezel overlay to support sensing of two fingers or styluses. With such line of sight systems, occlusion can be an issue for sensing.

The Lemur music controller from JazzMutant[11] uses a proprietary resistive overlay technology to track up to 20 touch points simultaneously. More recently, Balda AG and N-Trig[20] have both released capacitive multitouch overlays, which have been used in the iPhone and the Dell XT, respectively. These approaches provide a robust way for sensing multiple fingers touching the surface, but do not scale to whole hand sensing or tangible objects.

6.4. The need for intrinsically integrated sensing

The previous sections have presented a number of multitouch display technologies. Camera-based systems produce very rich data but have a number of drawbacks. Opaque sensing systems can more accurately detect fingers and objects, but by their nature rely on projection. Transparent overlays alleviate this projection requirement, but the fidelity of sensing is reduced. It is difficult, for example, to support sensing of fingertips, hands and objects.

A potential solution which addresses all of these requirements is a class of technologies that we refer to as "intrinsically integrated" sensing. The common approach behind these is to distribute sensing across the display surface, integrating the sensors with the display elements. Hudson[8] reports on a prototype 0.7" monochrome display where LED pixels double up as light sensors. By operating one pixel as a sensor while its neighbors are illuminated, it is possible to detect light reflected from a fingertip close to the display. The main drawbacks are the use of visible illuminant during sensing and practicalities of using LED-based displays. SensoLED uses a similar approach with

visible light, but this time based on polymer LEDs and photodiodes. A 1" diagonal sensing polymer display has been demonstrated.[2]

Planar[1] and Toshiba[24] were among the first to develop LCD prototypes with integrated visible light photosensors, which can detect the shadows resulting from fingertips or styluses on the display. The photosensors and associated signal processing circuitry are integrated directly onto the LCD substrate. To illuminate fingers and other objects, either an external light source is required—impacting on the profile of the system—or the screen must uniformly emit bright visible light—which in turn will disrupt the displayed image.

The motivation for ThinSight was to build on the concept of intrinsically integrated sensing. We have extended the work above using invisible (IR) illuminant to allow simultaneous display and sensing, building on current LCD and IR technologies to make prototyping practical in the near term. Another important aspect is support for much larger thin touch-sensitive displays than is provided by intrinsically integrated solutions to date, thereby making it more practical to prototype multitouch applications.

7. CONCLUSION

In this article we have described a new technique for optically sensing multiple objects, including fingertips, through thin form-factor displays. Optical sensing allows rich "camera-like" data to be captured by the display and this is processed using computer vision techniques. This supports new types of human computer interfaces that exploit zero-force multi-touch and tangible interaction on thin form-factor displays such as those described in Buxton.[3] We have shown how this technique can be integrated with off-the-shelf LCD technology, making such interaction techniques more practical and deployable in real-world settings.

We have many ideas for potential refinements to the ThinSight hardware, firmware, and PC software. In addition to such incremental improvements, we also believe that it will be possible to transition to an integrated "sensing and display" solution which will be much more straightforward and cheaper to manufacture. An obvious approach is to incorporate optical sensors directly onto the LCD backplane, and as reported earlier early prototypes in this area are beginning to emerge.[24] Alternatively, polymer photodiodes may be combined on the same substrate as polymer OLEDs[2] for a similar result. The big advantage of this approach is that an array of sensing elements can be combined with a display at very little incremental cost by simply adding "pixels that sense" in between the visible RGB display pixels. This would essentially augment a display with optical multitouch input "for free," enabling truly widespread adoption of this exciting technology.

research highlights

References

1. Abileah, A., Green, P. Optical sensors embedded within AMLCD panel: design and applications. In *Proceedings of EDT'07* (San Diego, California, August 04, 2007), ACM, New York, NY.
2. Börgi, L. et al. Optical proximity and touch sensors based on monolithically integrated polymer photodiodes and polymer LEDs. *Org. Electron. 7* (2006).
3. Buxton, B. Multi-Touch Systems That I Have Known and Loved (2007), http://www.billbuxton.com/multitouchOverview.html.
4. Dietz, P., Leigh, D. DiamondTouch: a multi-user touch technology. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (Orlando, FL, Nov. 11–14, 2001), UIST '01. ACM, NY, 219–226.
5. Han, J.Y. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology* (Seattle, WA, Oct. 23–26, 2005), UIST '05. ACM, NY, 115–118.
6. HP TouchSmart, http://www.hp.com/united-states/campaigns/touchsmart/.
7. Hodges, S., Izadi, S., Butler, A., Rrustemi, A., Buxton, B. ThinSight: Versatile multi-touch sensing for thin form-factor displays. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (Newport, RI, Oct. 07–10, 2007), UIST '07. ACM, NY, 259–268.
8. Hudson, S.E. 2004. Using light emitting diode arrays as touch-sensitive input and output devices. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (Santa Fe, NM, Oct. 24–27, 2004), UIST '04. ACM, NY, 287–290.
9. Izadi, S. et al. C-Slate: A multi-touch and object recognition system for remote collaboration using horizontal surfaces. In *IEEE Workshop on Horizontal Interactive Human-Computer Systems* (Rhode Island, Oct. 2007), IEEE Tabletop 2007. IEEE, 3–10.
10. Izadi, S. et al. Experiences with building a thin form-factor touch and tangible tabletop. In *IEEE Workshop on Horizontal Interactive Human-Computer Systems* (Amsterdam, Holland, Oct. 2008), Tabletop 2008, IEEE, 181–184.
11. JazzMutant Lemur. http://www.jazzmutant.com/lemur_overview.php.
12. Jordà, S., Geiger, G., Alonso, M., Kaltenbrunner, M. The reacTable: Exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction* (Baton Rouge, LA, Feb. 15–17, 2007). TEI '07. ACM, NY, 139–146.
13. Krueger, M. Artificial Reality 2. Addison-Wesley Professional (1991). ISBN 0-201-52260-8
14. Lee, S., Buxton, W., Smith, K.C. S. A multi-touch three dimensional touch-sensitive tablet. *SIGCHI Bull 16*, 4 (Apr. 1985), 21–25.
15. van Loenen, E. et al. Entertaible: a solution for social gaming experiences. In *Tangible Play Workshop, IUI Conference* (2007).
16. Loviscach, J. Two-finger input with a standard touch screen. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (Newport, RI, USA, October 07–10, 2007). UIST '07. ACM, New York, NY, 169–172.
17. Malik, S., Laszlo, J. Visual touchpad: A two-handed gestural input device. In *Proceedings of the 6th International Conference on Multimodal Interfaces* (State College, PA, Oct. 13–15, 2004), ICMI '04. ACM, NY, 289–296.
18. Matsushita, N., Rekimoto, J. 1997. HoloWall: Designing a finger, hand, body, and object sensitive wall. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (Banff, Alberta, Canada, Oct. 14–17, 1997), UIST '97. ACM, NY, 209–210.
19. Matsushita, N., Avatsuka, Y., Rekimoto, J. 2000. Dual touch: A two-handed interface for pen-based PDAs. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (San Diego, California, US, Nov. 06–08, 2000), UIST '00. ACM, New York, NY, 211–212.
20. N-trig, duo-touch sensor, http://www.n-trig.com/.
21. Rekimoto, J. SmartSkin: An infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Changing Our World, Changing Ourselves* (Minneapolis, MN, Apr. 20–25, 2002), CHI '02. ACM, NY, 113–120.
22. Smart Technologies, http://smarttech.com (2009).
23. Tactex Controls Inc. Array Sensors. http://www.tactex.com/products_array.php.
24. Matsushita, T. LCD with Finger Shadow Sensing, http://www3.toshiba.co.jp/tm_dsp/press/2005/05-09-29.htm.
25. Westerman, W. Hand Tracking, Finger Identification and Chordic Manipulation on a Multi-Touch Surface. PhD thesis, University of Delaware (1999).
26. Wellner, P. Interacting with Paper on the Digital Desk. *CACM 36*, 7 (1993) 86–96.
27. Wilson, A.D. TouchLight: An imaging touch screen and display for gesture-based interaction. In *Proceedings of the 6th International Conference on Multimodal Interfaces* (State College, PA, Oct. 13–15, 2004). ICMI '04. ACM, NY, 69–76.
28. Wilson, A.D. PlayAnywhere: A compact interactive tabletop projection-vision system. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology* (Seattle, WA, Oct. 23–26, 2005), UIST '05. ACM, NY, 83–92.

Shahram Izadi, Steve Hodges, Alex Butler, Darren West, Alban Rrustemi, Mike Molloy, and William Buxton ([shahrami, shodges, dab]@microsoft.com), Microsoft Research Cambridge, U.K.

Page 1685 of 1714

VIDEOPLACE--An Artificial Reality

Myron W. Krueger, Thomas Gionfriddo and Katrin Hinrichsen
Computer Science Department
University of Connecticut
Storrs, Connecticut 06268

Abstract

The human-machine interface is generalized
beyond traditional control devices to permit phy-
sical participation with graphic images. The
VIDEOPLACE System combines a participant's live
video image with a computer graphic world. It also
coordinates the behavior of graphic objects and
creatures so that they appear to react to the
movements of the participant's image in real-time.
A prototype system has been implemented and a
number of experiments with aesthetic and practical
implications have been conducted.

Introduction

This paper describes a number of experiments
in alternate modes of human-machine interaction.
The premise is that interaction is a central, not
peripheral, issue in computer science. We must
explore this domain for insight as well as immedi-
ate application. It is as important to suggest
new applications as it is to solve the problems
associated with existing ones. Research should an-
ticipate future practicality and not be bound by
the constraints of the present.

Unlike most computer science professionals,
who have been content to rely on traditional com-
puter languages and the hundred year old keyboard
as the means of input, designers of graphic sys-
tems have long recognized the importance of the
human-machine interface. Even so, most innova-
tions, including the light pen, joy stick, data
tablet and track ball have been dictated by the
minimum needs of immediate graphics applications.

There have been few experiments motivated by
a purely intellectual desire to explore the means
through which people and machines might interact,

independent of specific applications. One such
novel approach was Ivan Sutherland's head-mounted
stereo displays which sensed the orientation of
the viewer's head and displayed what would be seen
in a simulated graphic environment from each
position.[SUTH68]

Another unique approach was taken with the
GROPE system at the University of North Carolina.
It provided force feedback to a remote manipulator
that could be used to pick up graphic
blocks.[BATT72] In addition, there have been the
well funded efforts of the Architecture Machine
Group at MIT, including the Dataland project,
MovieMap and "Put that there".
[BOLT79, 80, 81], [LIPP80]

Finally, my work in Responsive Environments,
beginning in 1969 and continuing to the present,
has allowed a participant's movements around a
room to be translated into actions in a projected
graphic scene generated by the
computer.[KRUE77, 83]

This paper describes one of my early experi-
ments with Responsive Environments, the VIDEOPLACE
project currently under development and applica-
tions planned for the near future.

Abstract Versus Concrete Intelligence

The observation underlying this research is
that there are two quite different aspects of hu-
man intelligence. The first is the logical, deduc-
tive, explicitly rational process that we associ-
ate with abstract symbolic reasoning. While the
technically inclined take great pride in this
skill, a large fraction of the population has no
interest in developing it. The second is the fa-
cility for understanding, navigating and manipu-
lating the physical world. This ability is part of
our basic human heritage.

As a greater percentage of the population be-
comes involved in the use of computers, it is na-
tural to expect the manner of controlling comput-
ers to move away from the programming model and
closer to the perceptual process we use to accom-
plish our goals in the physical world.

Page 1686 of 1714

Early Responsive Environments

In 1969, I began to explore the idea of physical participation in a graphic world using the paradigm of a Responsive Environment. A Responsive Environment is an empty room in which a single participant's movements are perceived by the computer which responds through visual displays and electronic sound. Since 1970, video projection of computer graphic images has been used to provide the visual response.

PSYCHIC SPACE

In PSYCHIC SPACE, a Responsive Environment created in 1971, sensing of the participant's behavior was accomplished through a grid of hundreds of pressure sensors placed in the floor. As the participant walked around the room, the computer scanned the floor and detected the movement of his feet. The person's position in the room was then used to control an interaction in a graphic scene which was displayed on an 8'x10' rear-screen video projection.

In one PSYCHIC SPACE interaction, the participant's movements in the room were used to control the movements of a symbol on the video screen. After a few minutes, allocated for exploration of this phenomenon, a second symbol appeared. The participant, inevitably wondering what would happen if he walked his symbol over to the intruder's position, moved until the two symbols coincided. At that point, the second symbol disappeared and a maze appeared with the participant's symbol at the starting point. (Fig. 1)
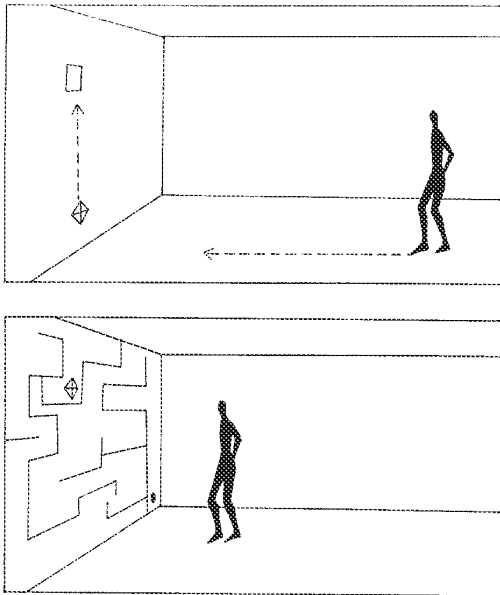

Fig. 1

Again, inevitably, the participant tried to walk the maze. However, after a few minutes the participant would realize that since there were no physical boundaries in the room, there was nothing to prevent cheating. When this realization struck, the participant, typically with some ceremony, raised his foot and planted it on the other side of one of the graphic boundaries. However, the maze program had anticipated this response and stretched that boundary elastically. Subsequent cheating attempts were greeted with a number of other gambits. The participant's symbol might fall apart; the whole maze could move; or, a specific boundary would disappear and a new one would appear elsewhere. By the end of the experience, the participant could have encountered as many as forty different variations on the maze theme.(KRUE77, 83)

PSYCHIC SPACE was presented as an aesthetic work in the Union Gallery at the University of Wisconsin. It suggested a new art form in which the participant's expectations about cause and effect could be used to create interesting and entertaining experiences, quite unlike anything that existed at that time and still different in spirit from the video games of today.

VIDEOPLACE

Concept

In 1970, I combined computer graphic images, created by an artist using a data tablet, with the live image of people. Observing their reactions to this computer graphic graffiti led to the formulation of the VIDEOPLACE concept.

VIDEOPLACE is a computer graphic environment in which the participant sees his or her live image projected on a video screen. It may be alone on the screen, or there may be images of other people at different locations. In addition, there may be graphic objects and creatures which interact with the participant's image.

When people see their image displayed with a graphic object, they feel a universal and irresistible desire to reach out and touch it. (Fig. 2) Furthermore, they expect the act of touching to affect the graphic world. By placing each participant against a neutral background, it is possible to digitize the image of his silhouette and to recognize the moment when it touches a graphic object. The system can then cause the object to move, apparently in response to the participant's touch.

It is also possible for the computer to analyze the participant's image and to alter its appearance on the screen. By either analog or digital techniques, the participant's image can be scaled and rotated and placed anywhere on the screen. Thus, in principle, the participant could climb graphic mountains, swim in graphic seas, or defy gravity and float around the screen. The potential for new forms of interaction within this model is very rich, with certain application as an art form, likely application in education and telecommunication, as well as arguable application for general human-machine interaction.
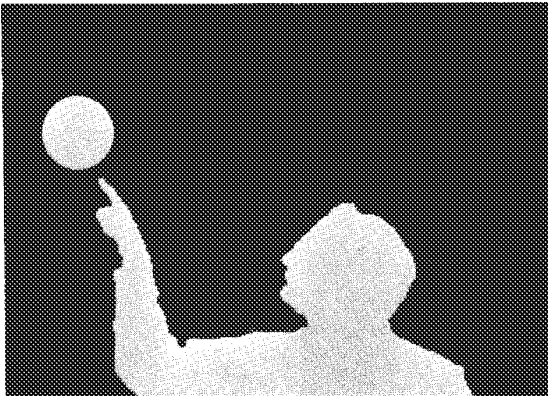
Page 1687 of 1714

Fig. 2

Prototype System

A prototype VIDEOPLACE system has been constructed. Since understanding the movements of the participant's image appeared to be the most challenging issue, much of the initial effort was focussed on solving this problem. Graceful mechanisms for specifying and controlling the desired interactive relationships have been developed. To date, only very simple graphics have been used because of the very modest resources available and the fact that until recently commercial equipment did not emphasize high speed manipulation of raster data. To a great extent, we have worked with graphic hardware of our own construction which provides a number of features important to our interactions. In addition, we have recently acquired three Silicon Graphics workstations, which will greatly enhance our ability to create and manipulate realistic three-dimensional scenes.

The software employed to control the interactions is quite unusual. We believe the methodology is of general interest for graphics and other real-time applications. We treat the overall system as a model of a real-time intelligence. It is divided into two major components: the Cognitive System which runs on a VAX11/780 and the Reflex System which consists of a group of closely coupled dedicated processors operating on a specialized bus structure.

The Reflex System handles instantaneous decision making. The plan is for the Cognitive System to monitor the events in the Environment and the decisions of the Reflex System, in order to understand what is happening in semantic terms and then to make strategic decisions that will alter the future character of the interaction.

Although all of the communications are established, the Cognitive System is not yet performing this monitoring function. However, it has totally altered the programming process. Instead of writing a separate program for each interaction, we describe the desired causal relationships in conceptual terms. This conceptual representation is then translated into a form that the Reflex System can interpret in real-time. The long term

objective is to develop an online real-time intelligence that understands the participant's behavior and the interaction in human terms.

CRITTER, A VIDEOPLACE Interaction

In one current interaction, the participant is joined by a single graphic creature on the screen. The behavior of this creature is very complex and context dependent. The intent is to produce the sensation of an intelligent and witty interaction between creature and the participant.

Initially, the creature sees the participant and chases his image about the screen. If the participant moves rapidly towards it, the creature, nicknamed CRITTER, moves to avoid contact. If the human holds out a hand, CRITTER will land on it and climb up the person's silhouette. As it climbs, its posture adapts to the contour of the human form. When it finally scales the person's head, it does a triumphant jig.

Once this immediate goal is reached, the creature considers the current orientation of the person's arms. If one of the hands is raised, it does a flying somersault and lands on that hand. If the hand is extended to the side but not above the horizontal, CRITTER dives off the head, rolls down the arm, grabs the finger and dangles from it. When the person shakes his hand, CRITTER falls off and dives to the bottom of the screen. Each time it climbs to the top of the participant's head, it is in a different state and is prepared to take a different set of actions. (Fig. 3a-h)

The CRITTER experience will soon be enhanced in a number of ways. Hardware has been built that shrinks the human image down to CRITTER size. The smaller size increases the number of relationships that can exist between the participant and the creature. Simple graphic scenes are being added. Both human and graphic entities will interact with these graphic props by moving among them, climbing them or hiding behind them. The new displays will provide a capability for three-dimensional scenery which can be navigated in real-time.

Practical Applications

The interface described is a deliberately informal one. The resemblance to video games might seem frivolous to the hard-nosed computer scientist used to catering to the needs of government agencies and three letter companies. However, games are a multi-billion dollar industry and by that measure practical. More importantly, games provide an extremely compelling interface whose advantages should be considered for more standard applications. Therefore, before adapting the techniques described to fit a more familiar practical context, we will examine their potential in the current VIDEOPLACE environment.

Computer Aided Instruction

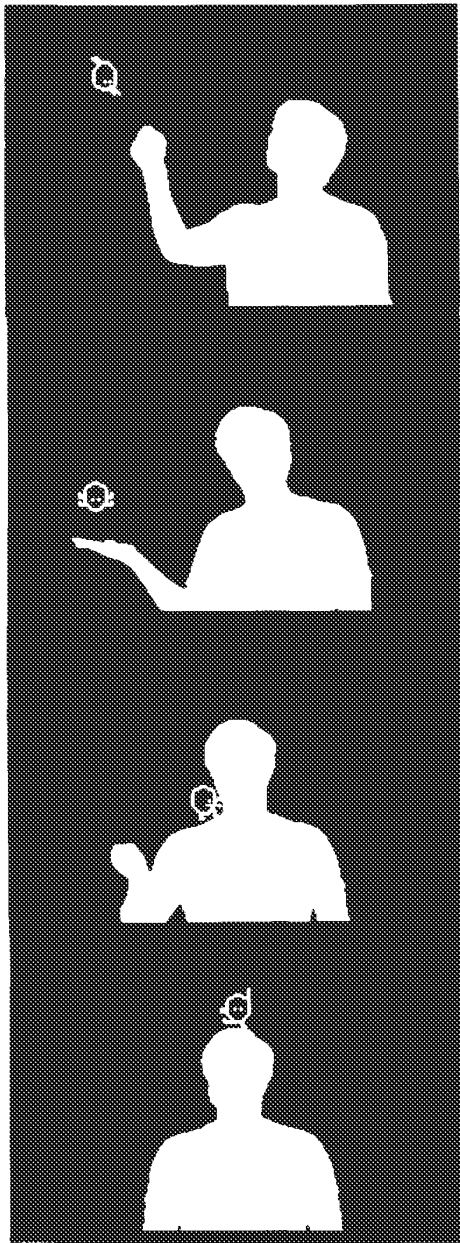In our culture, education is a sedentary activity imposed on naturally active creatures. Sti-

37

Fig. 3a-d

Fig. 3e-h

fling this energy is the first task of every elementary school teacher. As an alternative, VIDEO-PLACE could be used to create a physically active form of Computer Aided Instruction in which the computer is used not to teach traditional material, but to alter what, as well as how, we teach.

In one proposal, which I first made formally to NSF in 1975, elementary school children were to be placed in the role of scientists landing on an alien planet. VIDEOPLACE would be used to define an artificial reality in which the laws of cause and effect are composed by the programmer. The
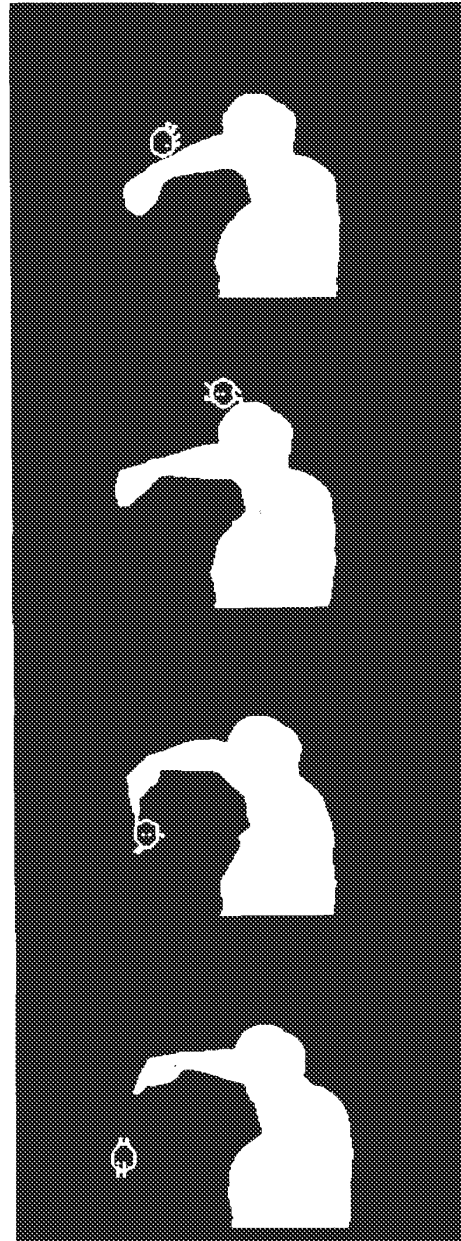
task of the children would be to discover these laws. They would enter the Environment singly, interact with it and make individual observations of its rules. Under the guidance of their teacher, they would discuss their experiences and present their opinions. They would compare notes and formulate theories. Since each child would behave differently, in the VIDEOPLACE, individuals would have unique experiences and produce conflicting theories. They would argue and then revisit the Environment, executing critical experiments to resolve which theories were correct under which conditions. Thus, students would learn how obser-

38

vation leads to hypothesis formation, prediction, testing and reformulation. They would learn the process of scientific thought rather than memorizing vocabulary and performing mechanical calculations as they often do now.

Telecommunications

VIDEOPLACE was originally conceived and implemented as a telecommunications environment allowing people in different places to share a common video experience. While the possibility of such graphic interaction may seem unnecessary to communication, we should remember two points. First, a hundred years ago the telephone seemed to have no advantage over the telegraph which could transmit the content of messages equally well. Second, since communication between friends or business associates is not limited to words, it is clearly desirable to provide a place in which individuals who are geographically separated can share a common visual environment.

An example of this use of VIDEOPLACE is described in _Artificial Reality_ (KRUE 83). A two-way computer graphic and live video telecommunications link was used to solve an engineering problem. In this experiment, the graphic images from two computers were viewed by television and combined by standard video techniques. Each participant pointed to the image on his local screen. The images of both of the participants' hands were combined with the graphic image, allowing them to gesture as naturally as if they were sitting together at a table. For the signal processing task at hand, the communication was complete.

Computing by Hand

A number of technologies are competing for space on the modern professional's desk. Telephones, answering devices, modems and computer terminals with touch screens are all candidates for the desk top. From the user's point of view, an empty desk is preferable. Two technology trends augur the removal of the computer terminal from the desk's surface. First, the keyboard will ultimately succumb to voice input. Second, flat screen displays of adequate resolution already exist. They are likely to be placed on a wall behind the desk, not on it, making touch screen input awkward.

The VIDEOPLACE techniques described in this paper can be used to duplicate any touch screen capability. A video camera pointed down at a desk surface can be used to create a VIDEODESK environment that will have several advantages over a touch screen.

In the VIDEOPLACE system, the user's hands can be used for any traditional graphics application. Since the system can detect when a person's hand touches a particular object, pointing and selection can be controlled. Similarly, a finger can be used to position the selected object in a design. A finger can also be used to draw on the screen, for example, to connect components in a logic design. We have already implemented simple menu selection, typing and finger painting systems. (Fig. 4)

Fig. 4

Video input offers more than a simple alternative to other pointing techniques. With the exception of the recent development of three-dimensional input devices, virtually all pointing devices are limited to two degrees of freedom. However, on the VIDEODESK, two hands can be used in concert to increase the user's bandwidth. In fact, in one common graphic application, it is easy to see the use for eight or more degrees of freedom. B-spline curves are used widely to design car bodies, ships hulls, turbine blades, etc. These curves are defined in terms of a relatively small number of control points. The user controls the shape of the curve by moving these points. With existing input devices only one point can be moved at a time. On the VIDEODESK, the tips of the index fingers and thumbs can be used to manipulate four control points simultaneously. (Fig. 5)

Conclusion

VIDEOPLACE is not so much a solution to existing problems, as an effort to stretch our thinking about the human-machine interface. We have already entered an era where most of the people using computers are no longer programmers in the traditional sense. We can look to a day when most of the people interacting with computers will not be users in the current sense.

Since computers are becoming less expensive than the people who use them, we can expect that as much computing power will be dedicated to providing a pleasing human-machine interface as is actually used to accomplish the user's application. As computer interaction becomes the dominant mode of performing work and transacting business, it becomes a significant ingredient in our quality of life. It is time to give the aesthetics of human-machine interaction serious thought.
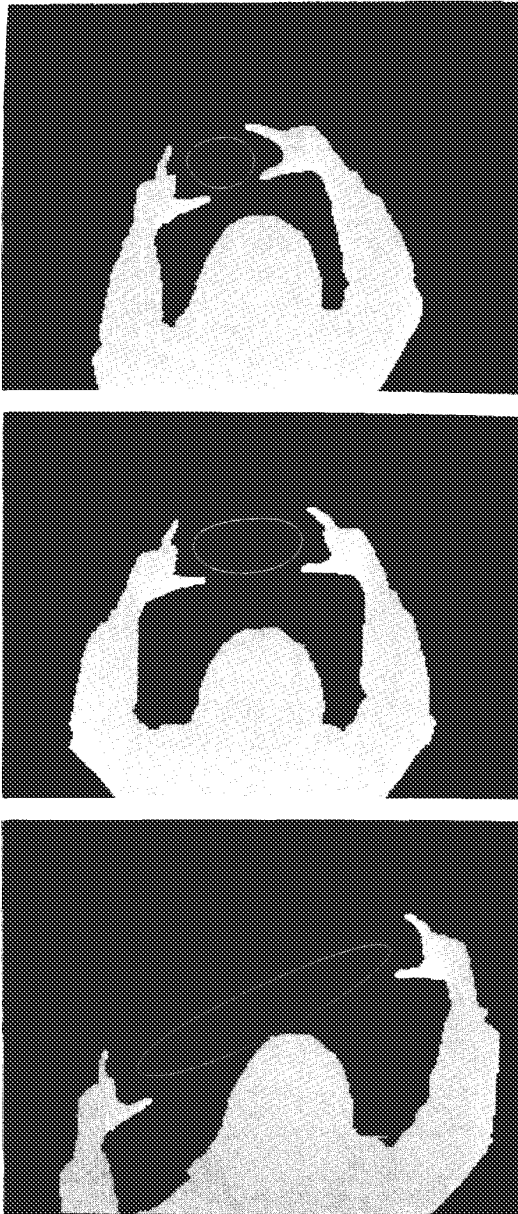
39

Fig. 5

Bibliography

[BATT72] Batter, James J. & Brooks, Frederick P. Jr., "GROPE-1," IFIPS 71, pp. 759-765.
[BOLT81] Bolt, Richard A., "Gaze Orchestrated Dynamic Windows." SIGGRAPH 81. pp. 32-42.
[BOLT80] Bolt, Richard A., "Put That There: Voice and Gesture at the Graphic Interfaces." SIGGRAPH 80. pp. 262-270.
[BOLT79] Bolt, Richard A., "Spatial Data Management." MIT 1979.
[CROC81] Crockett, D.W., "Triform Modules." CACM. Vol 24, No 6. pp. 344-350.
[CULL82] Cullingford, R.E., Krueger, M.W., Selfridge, M., "Automated Explanations in a CAD System." IEEE Transactions on SMC. January 1982.
[HAYE77] Hayes-Roth, F. and Lesser, V., "Focus of Attention in the Hearsay-II Speech Understanding System." IJCAI 77. pp. 27-35.
[KRUE83] Krueger, M.W., Artificial Reality. Addison-Wesley, 1983. 312 pp.
[KRUE81] Krueger, M.W., Cullingford, R.E. & Bellavance, D.A., "Control Issues in a CAD System with Expert Knowledge." Proceedings SMC. Oct. 1981.
[KRUE77] Krueger, M.W., "Responsive Environments." AFIPS 1977. 46:423-429.
[KRUE75] Krueger, M.W., IVAM, Annual Reports of NOAA Contract #5-315156 1975-1978.
[LIPP80] Lippman, Andrew. "Movie Maps: An Application of the Optical Disc to Computer Graphics." SIGGRAPH 81. pp. 109-120.
[SCHA77] Schank, R. and Abelson, R.P., "Scripts, Plans, Goals and Understanding." Erlbaum Press, 1977.
[SUTH68] Sutherland, Ivan. 1968. "A Head-Mounted Three-Dimensional Display." FJCC AFIPS 33-1:7575-764.

Brown, E., Buxton, W. & Murtagh, K. (1990). Windows on tablets as a means of achieving virtual input
devices. In D. Diaper et al. (Eds), *Human-Computer Interaction - INTERACT '90*. Amsterdam: Elsevier
Science Publishers B.V. (North-Holland), 675-681.

WINDOWS ON TABLETS AS A MEANS OF ACHIEVING VIRTUAL INPUT DEVICES

Ed BROWN, William A.S. BUXTON and Kevin MURTAGH

Computer Systems Research Institute,
University of Toronto,
Toronto, Ontario,
Canada M5S 1A4

Users of computer systems are often constrained by the limited number of physical devices at their
disposal. For displays, window systems have proven an effective way of addressing this problem.
As commonly used, a window system partitions a single physical display into a number of different
virtual displays. It is our objective to demonstrate that the model is also useful when applied to
input.

We show how the surface of a single input device, a tablet, can be partitioned into a number of
virtual input devices. The demonstration makes a number of important points. First, it demonstrates
that such usage can improve the power and flexibility of the user interfaces that we can implement
with a given set of resources. Second, it demonstrates a property of tablets that distinguishes them
from other input devices, such as mice. Third, it shows how the technique can be particularly
effective when implemented using a touch sensitive tablet. And finally, it describes the
implementation of a prototype an "input window manager" that greatly facilitates our ability to
develop user interfaces using the technique.

The research described has significant implications on direct manipulation interfaces, rapid
prototyping, tailorability, and user interface management systems.

1. INTRODUCTION

A significant trend in user interface design is away from the discrete, serial nature of what we might call a digital
approach, towards the continuous, spatial properties of an analogue approach.

Direct Manipulation systems are a good example of this trend. With such systems, controls and functions (such
as scroll bars, buttons, switches and potentiometers) are represented as graphical objects which can be thought
of as virtual devices. A number of these are illustrated in Fig. 1.
The impression is that of a number of distinct devices, each with its own specialized function, and occupying its
own dedicated space. While powerful, the impression is an illusion, since virtually all interactions with these
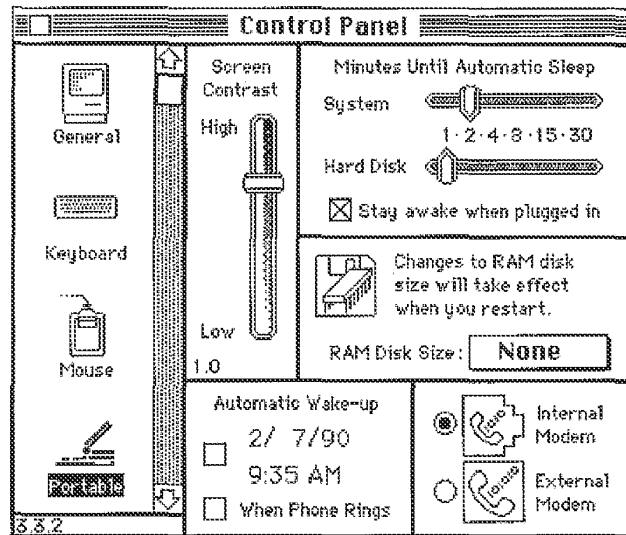devices is via only one or two physical devices: the keyboard and the mouse.

Page 1692 of 1714

Figure 1: Virtual Devices in the Macintosh Control Panel

The figure shows graphical objects such as potentiometers, radio buttons and icons. Each functions as a distinct device. Interaction, however, is via one of two physical devices: the mouse or keyboard.

The strength of the illusion, however, speaks well for its effectiveness. Nevertheless, this paper is rooted in a belief that direct manipulation systems can be improved by expanding the design space to better afford turning this illusion into reality. Distinct controls for specific functions, provide the potential to improve the directness of the user's access (such as through decreased homing time and exploiting motor memory). Input functions are moved from the display to the work surface, thereby freeing up valuable screen real-estate. Because they are dedicated, physical controls can be specialized to a particular function, thereby providing the possibility to improve the quality of the manipulation .

While one may agree with the general concepts being expressed, things generally break down when we try to put these ideas into practice. Given the number of different functions and virtual devices that are found in typical direct manipulation systems, having a separate physical controller for each would generally be unmanageable. Our desks (which are already crowded) would begin to look like an aircraft cockpit or a percussionist's studio. Clearly, the designer must be selective in what functions are assigned to dedicated controllers. But even then, the practical management of the resources remains a problem.

The contribution of the current research is to describe a way in which this approach to designing the control structures can be supported. To avoid the explosion of input transducers, we introduce the notion of virtual input devices that are spatially distinct. We do so by partitioning the surface of one physical device into a number of separate regions, each of which emulates the function of a separate controller. This is analogous on the input side to windows on displays.
We highlight the properties that are required of the input technology to support such windows, and discuss why certain types of touch tablets are particularly suited for this type of interaction.

Finally, we discuss the functionality that would be required by a user interface management system to support the approach. We do so by describing the implementation of a working prototype system.

Page 1693 of 1714

2. RELATIONSHIP TO PREVIOUS PRACTICE

The idea of virtual devices is not new. One of the most innovative approaches was the virtual keyboard developed by Ken Knowlton (1975, 1977a,b) at Bell Laboratories. Knowlton developed a system using half-silvered mirrors to permit the functionality of keyboards to be dynamically reconfigured. Partitioning a tablet surface into regions is also not new. Tablet mounted menus, as seen in many CAD systems, are one example of existing practice.

Our contribution:

- makes this model explicit
- develops it beyond current common practice
- develops some of the design issues (such as input transducers)
- demonstrates its utility
- and presents a prototype User Interface Management (UIMS) utility to support its use.

3. RELEVANT PROPERTIES OF INPUT TRANSDUCERS

The technique of "input windows" involves a mapping of different functions to distinct physical locations in the control space. This mapping can only be supported by input transducers that possess the following two properties:

- *Position Sensitive:* They must give absolute coordinates defining position, rather than a measure of motion (as with mice).
- *Fixed Planar Coordinate System:* Position must be measured in terms of a two dimensional Cartesian space.

Hence digitizing tablets will work, but mice, trackballs, and joysticks will not. Within the class of devices which meet these two criteria (including light pens, graphics tablets, touch screens), touch technologies (and especially touch tablets) have noteworthy potential.

Control systems that employ multiple input devices generally have two important properties:

- *Eyes-Free Operation:* Sufficient kinesthetic feedback is provided to permit the operation of the control, leaving the eyes free to perform some other task, such as monitoring a display.
- *Simultaneous Access:* More than one device can be operated at a time, as in driving a car (steering wheel and gear lever) or operating an audio mixing console (where multiple faders might be accessed simultaneously).

In many design situations, these properties are useful, if not essential. In mixing a colour in a paint program, one might assign a potentiometer to each of hue, saturation and value. In performing the task, it is reasonable to expect that the artist generally is better served by focusing visual attention on the colour produced rather than the potentiometers controlling its components values. Driving a car would be impossible if operating the steering wheel required visual attention.

Simultaneous access is also important in many situations. Within the domain of human-computer interaction, for example, Buxton and Myers (1986) demonstrate benefits in tasks similar to those demanded in text editing and

Page 1694 of 1714

CAD.

4. THE AFFORDANCES OF TOUCH TABLETS

Touch tablets are interesting in that they can be designed and employed in such a way as to afford eyes-free operation and simultaneous access. As well, they can meet our constraints of providing absolute position information in a planar coordinate system. In this, they are rare among input transducers.

The primary attribute of touch technologies that affords eyes-free operation is their having no intermediate hand-held transducer (such as a stylus or puck). Sensing is with the finger. Consequently, physical templates can be placed over a touch tablet (as illustrated in Fig. 2) and provide the same type of kinesthetic feedback that one obtains from the frets on a guitar or the cracks between the keys of a piano. This was demonstrated in Buxton, Hill and Rowley (1985). Because of the ability to memorize the position of virtual devices and sense their boundaries, usage is very different than that where a stylus is used, or where the virtual devices are delimited on the tablet surface graphically, and cannot be felt.

An interesting result from our studies, however, is the degree to which eyes-free control can be exercised on a touch tablet which is partitioned into a number of virtual devices, but which has no graphical or physical templates on the tablet surface.
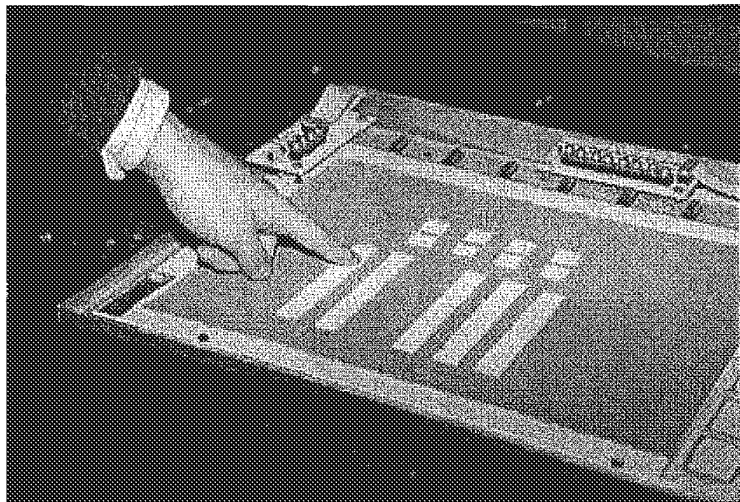
Figure 2: Using a template with a touch tablet

A cut-out template is being placed over a touch tablet. Each cut-out represents a different virtual device on a prototype operating console. The user can operate each device "eyes-free" since boundaries of the virtual devices can be felt (due to the raised edges of the template). If the tablet can sense more than one point of contact at a time, multiple virtual devices can be operated at once. (From Buxton, Hill, & Rowley, 1985).

Page 1695 of 1714

Figure 3: A 3"x3" Touch Tablet

A touch tablet of this size has the important property that it is on the same spatial scale as the hand. Therefore, control and access over its surface falls within the bounds of the relatively highly developed fine motor skills of the fingers, even if the palm is resting in a fixed (home?) position.

Using a 3"x3" touch sensitive touch tablet (shown in Fig. 3), our informal experience suggests that with very little training users can easily discriminate regions to a resolu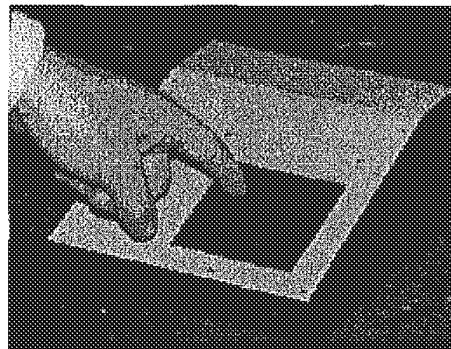tion of up to 1/3 of the tablet surface's vertical or horizontal dimensions. Thus, one can implement three virtual linear potentiometers by dividing the surface into three uniform sized rows or columns, or, for example, one can implement nine virtual push-button switches by partitioning the tablet surface into a 3x3 matrix.

If the surface is divided into smaller regions, such as a 4x4 grid, the result will be significantly more errors, and longer learning time. In such cases, using the virtual devices will require visual attention. The desired eyes-free operability is lost.

These limits are illustrated in Fig. 4. For example, we see that nine buttons for playing tick-tack-toe can work rather well, while a sixteen button numerical button keypad does not. Similarly, three virtual linear faders to control Hue, Saturation and Value work, while four such potentiometers do not.

Our belief is that the performance that we are observing is due to the size of the tablet as it relates to the size of the hand, and the degree of fine motor skills developed in the hand by virtue of everyday living. Being sensitive to these limits is very important as we shall see later when we discuss "dynamic windows." Because of this importance, these limits of motor control warrant more formal study.[1]
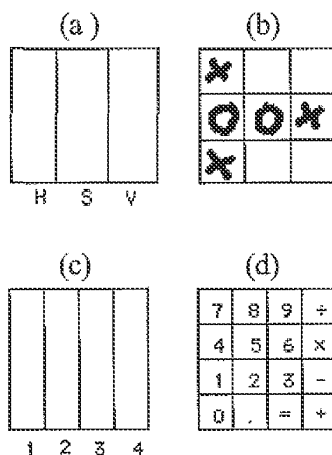
Page 1696 of 1714

Figure 4: Grids on Touch Tablets

*Four mappings of virtual devices are made onto a touch tablet. In (a) and (c), the regions
represent linear potentiometers. The surface is partitioned into 3 and 4 regions, respectively.
In (b) and (d) the surface is partitioned into a matrix of push buttons (3x3 and 4x4,
respectively). Using a 3"x3" touch tablet without templates, our informal experience is that
users can resolve virtual devices relatively easily, eyes-free, when the tablet is divided into
up to 3 regions in either or both dimensions. This is the situation illustrated in (a) and (b).
However, resolving virtual devices where the surface is more finely divided, as in (c) and (d),
presents considerably more load. Eyes-free operation requires far more training, and errors
are more frequent. The limits on this discrimination warrant more formal study.*

Finally, there is the issue of parallel access. Touch technologies have the potential to support multiple virtual
devices simultaneously. Again, this is largely by virtue of their not demanding any hand-held intermediate
transducer. If, for example, I am holding a stylus in my hand, the affordances of the device bias my expectations
towards wanting to draw only one line at a time. In contrast, if I were using finger paints, I would have no such
restrictive expectations.

A similar effect is at play in interacting with virtual devices implemented on touch tablets. Consider the template
shown in Fig. 2. Nothing biases the user against operating more than one of the virtual linear potentiometers at a
time. In fact, experience in the everyday world of such potentiometers would lead one to expect this to be
allowed. Consequently, if it is not allowed, the designer must pay particular attention to avoiding probable errors
that would result from this false expectation.

Being able to activate more than one virtual device at a time opens up a new possibilities in control and
prototyping. The mock up of instrument control consoles is just one example. The biggest obstacle restricting the
exploitation of this potential is the lack of a touch tablet that is capable of sensing multiple points. However, Lee,
Buxton and Smith (1985) have demonstrated a working prototype of such a transducer, and it is hoped that the
applications described in this current paper will help stimulate more activity in this direction.

In summary, we have seen that position sensitive planar devices readily support spatially distinct virtual input
devices. Further, we have seen that touch technologies, and touch tablets in particular, have affordances which
are particularly well suited to this type of interaction. Finally, it has been shown that a touch tablet capable of
sensing more than one point of contact at a time would enable the simultaneous operation of multiple virtual
devices.

5. VIRTUAL INPUT TRANSDUCERS

In current "menu on the tablet" practice, there is typically just one device driver which returns a single stream of
coordinates. The application must decode the data according to the current partitioning of the tablet. This is all ad
hoc, as are the means of specifying the boundaries of the various partitions. There are few tools, and little
flexibility.

In our approach, the data from each virtual device is transmitted to the application as if it were coming from an
independent physical device with its own driver. If the region is a button device, its driver transmits state changes.

Page 1697 of 1714

If it is a 1-D relative valuator, it transmits one dimension of relative data in stream mode. All of this is accomplished by placing a "window manager" between the device driver for the sensing transducer and the application.[2] Hence, applications can be constructed independent of how the virtual devices are implemented, thereby maintaining all of the desired properties of device independence. Furthermore, this is accomplished with a uniform set of tools that allows one to define the various regions and the operational behaviour of each region.

6. WHAT ABOUT DYNAMIC WINDOWS?

Window managers for displays can support the dynamic creation, manipulation, and destruction of windows. Is it reasonable to consider comparable functionality for input windows?

Our research (Buxton, Hill & Rowley, 1985) has demonstrated that under certain circumstances, the mapping of virtual devices onto the tablet surface can be dynamically altered. For example, in a paint system, the tablet may be a 2D pointing device in one context, and in another (such as when mixing colours) may have three linear potentiometers mapped onto it.

Changing the mapping of virtual devices onto the tablet surface restricts or precludes the use of physical templates. However, this is not always a problem. If visual (but not tactile) feedback is required, then a touch sensitive flat panel display can provide graphical feedback as to the current mapping. This is standard practice in many touch screen "soft machine" systems.

As has already been discussed, under certain circumstances, some touch tablets can be used effectively without physical or graphical templates. This can be illustrated using a paint mixing example. Since there are three components to colour, three linear potentiometers are used. As in Fig. 4(b), the potentiometers are vertically oriented so that there is no confusion: up is increase, down is decrease. The potentiometers are, left-to-right, Hue, Saturation, and Value (H, S & V in the figure). This ordering is consistent with the conventional order in speech, consequently there is little or no confusion for the user.

The example illustrates three conditions for using virtual devices without templates:

  * a low number of devices;
  * careful layout;
  * strong compatibility between the virtual devices and the application.

Our objective is not to encourage or legitimize the arbitrary use of menus on tablet surfaces. As many CAD systems illustrate, this often leads to bad user interface design. What we hope we have done is identify a technique which, when used in the appropriate context, will result in an improved user interface.

7. UIMS's AND VIRTUAL DEVICES

User Interface Management Systems, or UIMS's, are sets of tools designed to support iterative development of user interfaces through all phases of development (Tanner & Buxton, 1983; Buxton, Lamb, Sherman & Smith, 1983). Ideally, this includes specification, design, implementation, testing, evaluation and redesign. Typically, UIMS's provide tools for the layout of graphic interfaces, control low level details of input and output, and (more rarely) provide monitoring facilities to aid in evaluation of the interfaces developed.

Page 1698 of 1714

We have developed an *input window manager* (IWM). The tool consists of a "meta device" that provides for quick specification of the layout and behavior of the virtual devices. The specified configuration functions independent of the application. Users employ a *gesture-based trainer* to "show" the system the location and type of virtual device being specified. Hence, for example, adding a new template involves little more than tracing its outline on the control surface, defining the virtual device types and ranges, and attaching them to application parameters. Since the implementation of new devices can be achieved as quickly as they can be laid out on the tablet, this tool provides a new dimension of *system tailorability*.

In order to support iterative development, the tool should allow the user to suspend the application program, change the input configuration (by invoking a special process to control the virtual devices), and then proceed with the application program using the altered input configuration.

Property Sensed		Number of Dimensions							
		1		2				3	
Position	Rotary Pot	Sliding Pot	Tablet & Puck	Tablet & Stylus	Light Pen	Floating Joystick	3D Joystick	M
			+0	Touch Tablet +0		Touch Screen		+0	T
Motion	Continuous Rotary Pot	Treadmill	Mouse			Trackball	3D Trackball	M
		+0	Ferinstat +0	+0			X/Y Pad	+0	T
Pressure	Torque Sensor	+	+			Isometric Joystick	+	T

Figure 5. Taxonomy of Hand-Controlled Continuous Input Devices.

Cells represent input transducers with particular properties. Primary rows (solid lines) categorize property sensed (position, motion or pressure). Primary columns categorize number of dimensions transduced. Secondary rows (dashed lines) differentiate devices using a hand-held intermediate transducer (such as a puck or stylus) from those that respond directly to touch - the mediated (M) and touch (T) rows, respectively. Secondary columns group devices roughly by muscle groups employed, or the type of motor control used to operate the device. Cells marked with a "+" can be easily be emulated using virtual devices on a multi-touch tablet. Cells marked with a "O" indicate devices that have been emulated using a conventional digitizing tablet. After Buxton (1983).

8. THE REPERTOIRE OF SUPPORTED VIRTUAL DEVICES

The impact of the physical device used on the quality of interaction has been discussed by Buxton (1983). The objective, therefore, is to make available as broad a repertoire of "virtual" devices as possible from a limited number of physical transducers. We based our initial prototype on a conventional graphics tablet, and have designed to include future support for both single and multiple touch-sensitive tablets. The repertoire of virtual

Page 1699 of 1714

devices supported by our prototype is indicated in Fig 5.

9. A PROTOTYPE INPUT WINDOW MANAGER

The architecture of the IWM that we have implemented is depicted in Figure 6. The user interacts with the IWM at two separate points indicated by ovals in the diagram. The *Trainer* program, provides for configuring the input control structure. The *application* exists outside of the IWM, and the workings of the IWM are incidental to it (other than the interface to the request handler).
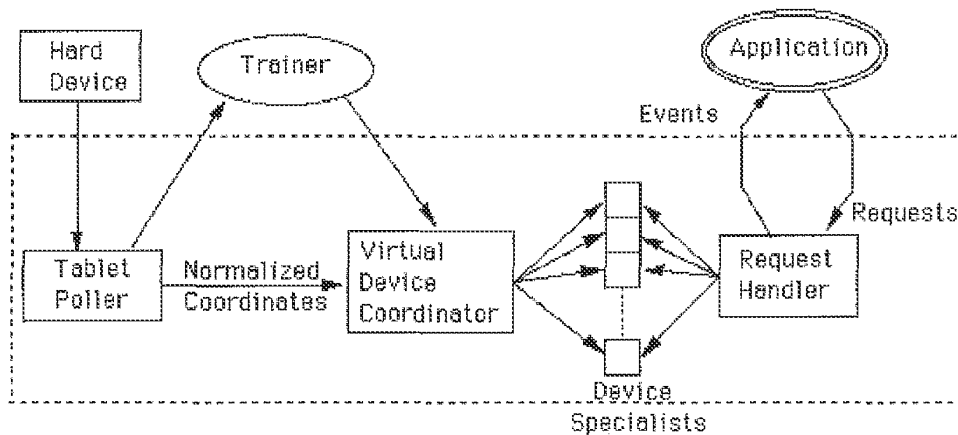

Figure 6. Architecture of a Prototype Input Window Manager

The *tablet poller* monitors the activity on the physical device, filters redundant information, and normalizes the data points before passing them on. The normalized format allows use of a range of physical devices simply by changing the tablet poller for the specific device.

The *virtual device coordinator* is active if the current activity is not a trainer session. It uses the incoming tablet data and the configuration provided by a trainer session to identify the virtual device to which the incoming data belongs. It passes the appropriate information on to the device specialist (device driver) for that virtual device. The *device specialist* determines the effect of the input and signals the *request handler* appropriately.

The virtual devices are accessed by the application program through two communication routines. One routine allows the activation and deactivation of various types of event signals. The other routine accesses the *event-queue*, returning the specifics of the last event to be signaled. A number of requests are available to the activation routine, including discrete status checks on a device, turning the device "on" or "off" for continuous event signaling, and a utility shutdown request.

The request handler module interprets and acts on requests from the application program, altering or extracting information of the device specialists as needed. It posts appropriate events to the event queue.

Finally, the architecture is such that much of the underlying software can reside in a dedicated processor, thereby freeing up resources on the machine running the main application. This includes the part of the tablet poller, the internal representation of the current mapping of the virtual devices onto the tablet, and the virtual device coordinator.

10. CONCLUSION

Page 1700 of 1714

This paper has discussed one way of making direct manipulation interfaces more direct and manipulation more effective. The general approach has been to extend the number of discrete and continuous controllers which can be tied to different functions. This is accomplished through spatially distinct virtual devices, and an input window management system. In the process, a number of properties of input devices have been discussed, and a prototype system presented. The results have important implications on the usability and tailorability of systems, and the architecture of UIMS's.

The work described has been exploratory. Nevertheless, we feel that the results are sufficiently compelling to suggest that more formal investigations of the issues discussed are warranted. We hope that the current work will help serve as a catalyst to such research.

ACKNOWLEDGEMENTS

NOTES

1 It must be emphasized that the limits discussed here were obtained through informal study. We intend only to suggest that there is something interesting and useful here, rather than to imply that these are experimentally derived data.

2 We thank Alain Fournier for first suggesting the analogy with window managers.

REFERENCES

Anson, E. (1982). The Device Model of Interaction. Computer Graphics, (16,3), 107-114.

Buxton, W. (1983). Lexical and Pragmatic Considerations of Input Structures. Computer Graphics, (17,1), 31-37

Buxton W., Hill R., & Rowley P. (1985). Issues and Techniques in Touch-Sensitive Tablet Input. Computer Graphics, 19(3), 215 - 224.

Buxton, W., Lamb, M., Sherman, D., & Smith, K.C. (1982). Towards a Comprehensive User Interface Management System. Computer Graphics, (16,3), 99-106.

Buxton, W. & Myers, B. (1986). A Study in Two-Handed Input. Proceedings of CHI'86 Conference on Human Factors in Computing Systems, 321-326.

Evans, K. Tanner, P., & Wein, M. (1981). Tablet-based Valuators that Provide One, Two, or Three Degrees of Freedom. Computer Graphics (15,3), 91-97.

Kasik, D. (1982). A User Interface Management System. Computer Graphics, (16,3), 99-106.

Page 1701 of 1714

Knowlton, K. (1975). Virtual Pushbuttons as a Means of Person- Machine Interaction. Proc. IEEE Conf. on Computer Graphics, Pattern Matching, and Data Structure., 350-351.

Knowlton, K. (1977a). Computer Displays Optically Superimposed on Input Devices. The Bell System Technical Journal (56,3), 367-383.

Knowlton, K. (1977b). Prototype for a Flexible Telephone Operator's Console Using Computer Graphics. 16mm film, Bell Labs, Murray Hill, NJ.

Lee S., Buxton, W., & Smith, K.C. (1985). A Multi-Touch Three Dimensional Touch Tablet. Proceedings of CHI'85 Conference on Human Factors in Computing Systems, 21 - 25.

Myers, B. (1984a), Strategies for Creating an Easy to Use Window Manager with Icons. Proceedings of Graphics Interface '84, Ottawa, May, 1984, 227 - 233.

Myers, B. (1984b), The User Interface for Sapphire. IEEE Computer Graphics and Applications, 4 (12), 13 - 23.

Pike, R. (1983). Graphics in Overlapping Bitmap Layers. Computer Graphics, 17 (3), 331 - 356.

Tanner, P.P. & Buxton, W. (1985). Some Issues in Future User Interface Management System (UIMS) Development. In Pfaff, G. (Ed.), User Interface Management Systems, Berlin: Springer Verlag, 67 - 79.

Page 1702 of 1714

A Multi-Touch Three Dimensional Touch-Tablet

Electronic Patent Application Fee Transmittal

Application Number:	
Filing Date:	
Title of Invention:	Capacitive Responsive Electronic Switching Circuit
First Named Inventor/Applicant Name:	Byron Hourmand
Filer:	Brian A. Carlson
Attorney Docket Number:	5796183RX2

Filed as Small Entity

ex parte reexam Filing Fees

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Basic Filing:				
REQUEST FOR EX PARTE REEXAMINATION	2812	1	6000	6000
Pages:				
Claims:				
Reexamination Independent Claims	2821	5	210	1050
REEXAMINATION CLAIMS IN EXCESS OF TWENTY	2822	65	40	2600
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD($)
Post-Allowance-and-Post-Issuance:				
Extension-of-Time:				
Miscellaneous:				
			Total in USD ($)	9650

Electronic Acknowledgement Receipt

EFS ID:	17754459
Application Number:	90013106
International Application Number:	
Confirmation Number:	9188
Title of Invention:	Capacitive Responsive Electronic Switching Circuit
First Named Inventor/Applicant Name:	Byron Hourmand
Customer Number:	25962
Filer:	Brian A. Carlson
Filer Authorized By:	
Attorney Docket Number:	5796183RX2
Receipt Date:	24-DEC-2013
Filing Date:	
Time Stamp:	15:03:12
Application Type:	Reexam (Patent Owner)

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$9650
RAM confirmation Number	1558
Deposit Account	501065
Authorized User	
The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:	
Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees)	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Transmittal of New Application	PTO_NAR-5796183RX2_Reexam_Request_Form_PTO.pdf	125658 / f7f01010949a811011ceaadb8c6aeb2ec4c2c095	no	3
Warnings:					
Information:					
2	Receipt of Original Ex Parte Reexam Request	PTO_NAR-5796183RX2_Request_for_Reexam_PTO.pdf	350144 / e4cb2e4cddcd654c0466c77053e629754043d421	no	34
Warnings:					
Information:					
3	Reexam Miscellaneous Incoming Letter	PTO_NAR-5796183RX2_Exhibit_A_PTO2.pdf	6723309 / a60672b73e9aa2bdb0a60c5ee3b11f12caa63bb5	no	37
Warnings:					
Information:					
4	Reexam Miscellaneous Incoming Letter	PTO_NAR-5796183RX2_Exhibit_B_PTO2.pdf	17839163 / 5d37c96622cb2804ea03b3b41d6022278b5f07ba	no	136
Warnings:					
Information:					
5	Reexam Miscellaneous Incoming Letter	PTO_NAR-5796183RX2_Exhibit_C_PTO2.pdf	2031975 / 164bc7c660d83257e8969243352c131ad2e385f0	no	13
Warnings:					
Information:					
6	Reexam Miscellaneous Incoming Letter	PTO_NAR-5796183RX2_Exhibit_D_PTO2.pdf	2558063 / 6b2c2a6bc334b4e355e5cbc6c24bc59df7711fa2	no	16
Warnings:					
Information:					
7	Reexam Miscellaneous Incoming Letter	PTO_NAR-5796183RX2_Exhibit_E_PTO2.pdf	309214 / 6d597cfba861476043777049d6835394e24262dc	no	2
Warnings:					
Information:					
8	Preliminary Amendment	PTO_NAR-5796183RX2_Amend_Acc_Reexam_Req_PTO.pdf	683580 / 179abf2337afcc80e41cd06652a3000e163f9d8a	no	142
Warnings:					
Information:					

9	Copy of patent for which reexamination is requested	PTO_NAR-5796183RX2_183Patent_PTO.PDF	2272260 4f5329791f53446bb65436b84cf199105373e181	no	36
Warnings:					
Information:					
10	Assignee showing of ownership per 37 CFR 3.73.	PTO_NAR-5796183RX2_373_Form_PTO.pdf	123283 128c4c53bfe9c2b814cf0c633f8402e2330c74ce	no	3
Warnings:					
Information:					
11	Power of Attorney	PTO_NAR-5796183RX2_Power_of_Atty_PTO.PDF	302528 b07c6a63a6c63630476018b064dc44f4d895b527	no	1
Warnings:					
Information:					
12	Transmittal Letter	PTO_NAR-5796183RX2_IDS_Cover_Ltr_PTO.pdf	15259 c305711d1f9f296a4b83e31e8170fb106c6f7957	no	1
Warnings:					
Information:					
13	Information Disclosure Statement (IDS) Form (SB08)	PTO_NAR-5796183RX2_SB08_Form_PTO.pdf	695955 883bc66b5537b8497a0e5f97f27f0062d0ebba5d	no	7
Warnings:					
Information:					
14	Non Patent Literature	PTO_NPL_Buxton_InvitedPaper_PTO.pdf	409495 312f0417fac8701faeee3f50d9fa0e57b54f51b4	no	5
Warnings:					
Information:					
15	Non Patent Literature	PTO_NPL_Hinckley_PTO.pdf	248635 55da6e593e3e6b72c4b4f769a1e91ea0307c3ca2	no	4
Warnings:					
Information:					
16	Non Patent Literature	PTO_NPL_Lee_Thesis_PTO.pdf	16890598 ab9acf8a342960946582eb3e3b7206f9fbc6e5d8	no	118
Warnings:					
Information:					
17	Non Patent Literature	PTO_NPL_Hillis_AHighRes_PTO.pdf	1879168 aa40075463062ed7a2ebde71f2f34ec049f517ef	no	12
Warnings:					
Information:					

| 18 | Non Patent Literature | PTO_NPL_Lee_AMulti-Touch_PTO.pdf | 874510 | no | 5 |
| | | | df53aec5188e8773b1722ba895f8184e3172a8fc | | |

Warnings:

Information:

| 19 | Non Patent Literature | PTO_NPL_Hlady_PTO.pdf | 2257242 | no | 7 |
| | | | e0d2386694a86e80874a7684a55c3ae92dbfa5f2 | | |

Warnings:

Information:

| 20 | Non Patent Literature | PTO_NPL_Sasaki_PTO.pdf | 5118466 | no | 5 |
| | | | ce673fbb29fbbde7cffba06f5f0337047b2307cb | | |

Warnings:

Information:

| 21 | Non Patent Literature | PTO_NPL_Callahan_PTO.pdf | 1192812 | no | 6 |
| | | | adee8f6a5fb28823fc4295ececcb97cfe4790778 | | |

Warnings:

Information:

| 22 | Non Patent Literature | PTO_NPL_Casio_Ad_PTO.pdf | 321343 | no | 1 |
| | | | bab17c766e91da884c0f3235447ae2f22270bcb4 | | |

Warnings:

Information:

| 23 | Non Patent Literature | PTO_NPL_Casio_Manual_PTO.pdf | 1939711 | no | 14 |
| | | | b598ae16cb083a0d9b7db30bf8e478164238871c | | |

Warnings:

Information:

| 24 | Non Patent Literature | PTO_NPL_Smith_Bit-slice_PTO.pdf | 1363333 | no | 7 |
| | | | d82b4cdfb5dea978274a51afe2da349f570da041 | | |

Warnings:

Information:

| 25 | Non Patent Literature | PTO_NPL_Boie_PTO.pdf | 1522073 | no | 9 |
| | | | 574b22e97a3475322299d25a826d128ec602dae4 | | |

Warnings:

Information:

| 26 | Non Patent Literature | PTO_NPL_CliveThomson_PTO.pdf | 219994 | no | 3 |
| | | | 2c5bd016bd4ef378eccdac8ac2c124ae3efdfca6 | | |

Warnings:

Information:

27	Non Patent Literature	PTO_NPL_National_Research_Council_PTO.pdf	830527 f8ce9991051b72736e5e6b0797e3c0a4eee216ff	no	85
Warnings:					
Information:					
28	Non Patent Literature	PTO_NPL_Buxton_IssuesTech_PTO.pdf	2176107 f4d1d89e7255ef9edbf37a4c3fd385507f4bbc5d	no	10
Warnings:					
Information:					
29	Non Patent Literature	PTO_NPL_Buxton_LargeDisplays_PTO.pdf	699993 1aface304e3752dc2b700ebdd3d6f104dd26235d	no	8
Warnings:					
Information:					
30	Non Patent Literature	PTO_NPL_Buxton_LexicalPrag_PTO.pdf	1531125 781e65ebd8d45a45f6a0e30d1b5a5a61a24fc7a1	no	7
Warnings:					
Information:					
31	Non Patent Literature	PTO_NPL_LightBeam_PTO.pdf	81930 5027e8f24277c185a160634ffe254a811889e25b	no	2
Warnings:					
Information:					
32	Non Patent Literature	PTO_NPL_Buxton_Multi-Touch_PTO.pdf	643791 ee40592d58cadcfb1894732afd286d0035b0df7d	no	22
Warnings:					
Information:					
33	Non Patent Literature	PTO_NPL_Herot_PTO.pdf	1550938 87b7288c423780d930b4a5c062013ee322bd3249	no	7
Warnings:					
Information:					
34	Non Patent Literature	PTO_NPL_Wolfeld_PTO.pdf	15411551 619e70b30024524a72a6181ac4f6317a0334bb9e	no	68
Warnings:					
Information:					
35	Non Patent Literature	PTO_NPL_Lewis_PTO.pdf	613852 ecc53097f693967b592bad1026125a16ec91dec5	no	6
Warnings:					
Information:					

36	Non Patent Literature	PTO_NPL_Nakatani_PTO.pdf	1112483 5c2d9a7ff7d21d354e8097017e0608c9aa16 f7aa	no	5
Warnings:					
Information:					
37	Non Patent Literature	PTO_NPL_Rubine_PTO.pdf	3162925 557a95160c204d5233ff036d96b23bc10b8f 680d	no	285
Warnings:					
Information:					
38	Non Patent Literature	PTO_NPL_Kurtenbach_PTO.pdf	1670439 0dc0ca4099f4b05bb0bdf6025f87a999e2de 6e14	no	201
Warnings:					
Information:					
39	Non Patent Literature	PTO_NPL_Hopkins_PTO.pdf	113021 1967f1f1645e82db2843bf6c0557227911ed e095	no	8
Warnings:					
Information:					
40	Non Patent Literature	PTO_NPL_Buxton_LongNose_PTO.pdf	27893 621d2f1a416496295bdcf582e2595a3bb4c b29eb	no	3
Warnings:					
Information:					
41	Non Patent Literature	PTO_NPL_Buxton_MadDash_PTO.pdf	37401 d513532e5b9fc379095305b223f5f487e34d 0055	no	3
Warnings:					
Information:					
42	Non Patent Literature	PTO_NPL_NASA_Graphic_Manipulator_PTO.pdf	4547824 87aa779ba0b2789b7b017d0a92784dbd0b 59ad9d	no	28
Warnings:					
Information:					
43	Non Patent Literature	PTO_NPL_Izadi_PTO.pdf	1221374 ab87fdb52a4317631f99f54206dd79f0f187 ce87	no	9
Warnings:					
Information:					
44	Non Patent Literature	PTO_NPL_Krueger_PTO.pdf	1080548 c5868e5addea39874c8a8b01c14e5be5fb1 e35f7	no	6
Warnings:					
Information:					

45	Non Patent Literature	PTO_NPL_Brown_PTO.pdf	1150381 7cd9999fde19f7f57a96158c178b3277a5930bc4	no	11

Warnings:

Information:

46	Non Patent Literature	PTO_NPL_YouTube_Video1_PTO.pdf	478495 8caeaa56b47adc32cceca7a37f5062e1458b2656	no	1

Warnings:

Information:

47	Non Patent Literature	PTO_NPL_YouTube_Video2_PTO.pdf	611272 155f817a147f94608c33dc9f0a5e86b94823d38c	no	1

Warnings:

Information:

48	Fee Worksheet (SB06)	fee-info.pdf	33467 2041622c6292997365ba41e75802557ae353b4e6	no	2

Warnings:

Information:

Total Files Size (in bytes):	107055108

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Bib Data Sheet

CONFIRMATION NO. 9188

SERIAL NUMBER 90/013,106	FILING OR 371(c) DATE 12/24/2013 RULE	CLASS 307	GROUP ART UNIT 3992	ATTORNEY DOCKET NO. 5796183RX2

AIA (First Inventor to File): YES

INVENTORS
   5796183, Residence Not Provided;
   NARTRON CORPORATION, REED CITY, MI;

APPLICANTS
   5796183, Residence Not Provided;
   NARTRON CORPORATION, REED CITY, MI;

** CONTINUING DATA *************************
   This application is a REX of 08/601,268 01/31/1996 PAT 5796183

** FOREIGN APPLICATIONS *******************

** SMALL ENTITY **

Foreign Priority claimed ☐ yes ☐ no 35 USC 119 (a-d) conditions met ☐ yes ☐ no ☐ Met after Allowance Verified and Acknowledged _____ Examiner's Signature _____ Initials	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS 32	INDEPENDENT CLAIMS 8

ADDRESS
25962

TITLE
Capacitive Responsive Electronic Switching Circuit

FILING FEE RECEIVED 6000	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:	☐ All Fees
		☐ 1.16 Fees (Filing)
		☐ 1.17 Fees (Processing Ext. of time)
		☐ 1.18 Fees (Issue)
		☐ Other _____
		☐ Credit