ISSUES IN COMBINING MARKING AND DIRECT MANIPULATION TECHNIQUES

Gordon Kurtenbach and William Buxton

Computer Systems Research Institute,

University of Toronto,

Toronto, Canada

ABSTRACT

The direct manipulation paradigm has been effective in helping designers create easy to use mouse and keyboard based interfaces. The development of flat display surfaces and transparent tablets are now making possible interfaces where a user can write directly on the screen using a special stylus. The intention of these types of interfaces is to exploit user's existing handwriting, mark-up and drawing skills while also providing the benefits of direct manipulation. This paper reports on a test bed program which we are using for exploring hand-marking types of interactions and their integration with direct manipulation interactions.

Keywords: markings, gestures, stylus, pen-based interfaces, direct manipulation

1. INTRODUCTION

A popular vision of the future is a portable computer which allows one to write directly on the screen (Carr, 1991; Leitch, 1990; Normile & Johnson, 1990; Rebello, 1990). One can envision an interface where a stylus serves as the main input device supplying text commands and positioning information. A user interacts with the system as if it were pencil and paper. However, because real paper is not an active computational device, the pencil and paper metaphor is not sufficient by itself. In computer-based systems, handwriting may magically turn into formatted text, markings may cause an object to change state and objects may be pointed to and dragged. There are therefore many design issues which affect the ease of use and efficiency of these interfaces that need to be addressed. What do users expect? How do they know what to write and when? When and how do they use direct manipulation?

We use the term "marking interactions" to describe interactions where the pointing device leaves an "ink trail" on the display similar to writing with a pen. An example of an interface that uses marking interactions is a prototype spread-sheet described in Wolf, Rhyne & Ellozy (1989). Motivating our work is the view that marking interactions can be combined with *direct manipulation interfaces* (Shneiderman, 1982; Hutchins, Hollan & Norman, 1986) to produce easier to use and expressive interfaces. This view has also been expressed by Wolf and Rhyne (1987). The advantages of marking interfaces can be used to support the weaknesses of direct manipulation and vice versa. The PenPoint system by the Go Corporation is an example of a system which supports both marking interactions and direct manipulation (Carr, 1991).

1.1 GEdit

In order to explore this hybrid marking/direct manipulation design space we have created a program called *GEdit*. GEdit[1] is a prototype graphical editor that permits a user to create and manipulate three types of objects (squares, circles and triangles) using shorthand and proof-reader style markings. Using hand-drawn symbols, a user can add, delete, move and copy these objects. Objects can be manipulated either individually or in groups. Groups are specified with hand-drawn circling symbols.

This particular toy world was chosen because it is simple enough to be tractable, yet complex enough to be applicable to a set of real-world problems (such as one might encounter in a CAD or graphical layout program). The hope is that it will also provide a useful educational tool for those interested in character and gesture

[1]GEdit builds upon two earlier studies that investigated the use of similar markings and objects (Buxton, 1982; Buxton, Fiume, Hill, Lee & Woo, 1983). An earlier version of GEdit is described in Kurtenbach & Buxton (1990). The current version of GEdit differs from these earlier systems in that complete commands can be accomplished with single contiguous markings, smooth transitions from marking to direct manipulation are provided and a unique interactive way of helping users learn and remember the shorthand marks used to create objects.

recognition, and in integrating interaction techniques.[2] We have been using GEdit to explore various design issues and to demonstrate a number of unique interaction techniques. GEdit demonstrates the following:

• The use of proof-reader and shorthand markings to invoke commands;

• That different aspects of a mark can be used to control different command parameters (e.g, location, shape, size of the mark);

• That commands that require more than one argument can sometimes be effectively expressed in a single contiguous marking. For example, a typical proof-reader's "move" mark embeds three different pieces of information: the command (move), what is to be moved (direct object), and where it is to be moved to (indirect object). GEdit accomplishes the expression of complex commands in a similar fashion;

• That markings can eliminate the need for some kinds of modes (such as the need to select one tool at a time from a pallete). Thus many of the problems associated with modes can be avoided (such as mode errors, see Sellen, Kurtenbach, and Buxton, 1990);

• That by combining markings and direct manipulation, variations on a command invoked by an action can be achieved by varying some minor aspect of the action. This can be accomplished in a simple and consistent manner. (We will later illustrate this concept in the discussion of the "move" and "copy" commands.)

• That marking and direct manipulation techniques can be combined in a single interaction.

GEdit demonstrates several innovative interaction techniques and brings to light many issues relevant to the design of pen-based interfaces. The next section describes and discusses the techniques and issues in more detail.

2. GEDIT: INTERACTIONS AND DESIGN ISSUES

One interacts with GEdit using a number of simple markings. These are articulated using a pointing device, such as a stylus or a mouse. Commands are entered by moving the pointing device while maintaining downward pressure on the stylus or holding down the mouse button. Most commands leave an "ink trail" along the path of motion.

[2] GEdit runs on any Apple Macintosh computer. A version for the Sun (Suntools and X11) also exists. The program is available from the authors as shareware.

2.1 Creating Objects

Objects are created using a form of shorthand notation. Three objects can be defined: a square, circle and triangle. Shorthand symbols are used to define which type of object is to be created, and where it is to be placed (see Figure 1). Object type is defined by the shape and direction of the shorthand symbol. The new object is centered on the starting point of the defining symbol.

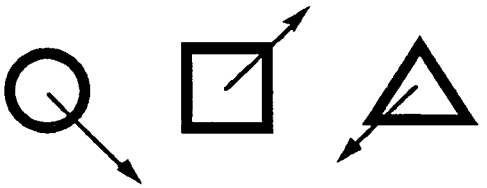


Figure 1. Shorthand symbols for adding objects. The objects (in bold) are created by the corresponding marks (thin lines with arrowheads).

This method of creating objects has several advantages over traditional direct manipulation techniques for creating objects. Typically in a direct manipulation system a user operates on an graphical object such as an icon which represents a file or directory. User actions affect the object only when the pointing device is within the area of the object, or when the object is selected (by clicking on it for example). There are several problems with this approach:

• How does one operate on objects not on the screen (the What You See is All You Get syndrome)?

• Most of the time one wants to create *and* place an object. Often, direct manipulation interfaces require two sets of actions to accomplish this.

• How does one create an object? If it does not exist, what object does one operate on to create it?

Generally a "creator object" can be manipulated to generate a new object. For example, consider creating a circle in MacDraw. First one must identify the "create circle tool" within the pallete, select it, place the circle and finally discard the tool or get a new one. This interface costs both in terms of screen real-estate and user action. More specifically, menu or pallete-based tool selection of this kind is highly moded. The overhead is both the time and actions required to switch modes, and the potential for mode errors to occur.

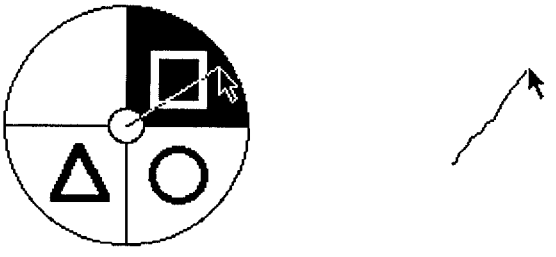
In GEdit there is no notion of "current tool" thus avoiding the problem of switching modes. A user tells the system what to create by using short-hand symbols rather than by using a special creator tool. Other advantages include the fact that no screen real-estate is consumed by "creator objects" and the creation command is combined with the placement command (drawing the symbol for the object

also determines where it is placed). When using GEdit it is apparent that object creation/layout is extremely efficient.

However, this technique has some tradeoffs. First, how does one know what mark to make to create an object or how does one find out? The problem is that markings are not *self-revealing*. When the system itself supplies information on what commands are available and how to invoke them through the mechanism used to invoke commands, we refer to this as being self-revealing. Menus and buttons, for example, are self-revealing: the set of available commands is readily visible as a by-product of the way commands are invoked. In contrast, markings are not intrinsically self-revealing.

2.2 Combining Menus and Markings

GEdit provides a technique which makes the short-hand marks for adding objects self-revealing. If a user is unsure of what markings can be made, the user presses down on the stylus and waits for a short interval of time. When the system detects that no mark is being made it prompts the user with a subdivided circular menu or "pie menu" (Callahan et al., 1988) of the three objects, appearing directly under the cursor.[1] The user may then select an object from the pie menu by keeping the stylus tip depressed and making a stroke through the desired sector or slice of the pie (Figure 2a). The slice is highlighted and the selection is confirmed when the pressure on the stylus is released. The object is then created and placed centered at the point where the stylus was first pressed down. A user can also indicate "no selection" by moving the stylus tip back to the center of the menu before releasing, or change selection by moving the tip to highlight another slice before releasing.



Prompted Selection (a)

Blind Selection (b)

Figure 2. The transition from novice to expert reflected in different ways of invoking commands.

The first important point to note is that *the physical movement involved in selecting an object is identical to the physical movement required to make the mark corresponding to that object.* For example, the square which requires an up and to the right movement for selection from the pie menu, also requires an up and to the right marking to create it.

The second point to note is that *supporting markings with pie menus in this way helps users make a smooth transition from novice to expert.* Novices in effect perform "menu selection". We have observed in the laboratory that users almost always wait for the pop-up menu and then select the desired object (Figure 2a) when they first encounter a new menu. However, waiting for the menu takes time, and thus as users begin to memorize the layout (as they become expert), they begin to "mark ahead" to create objects instead (Figure 2b). We have also observed an intermediate stage where users may begin to make a mark, and then wait for the menu to pop-up in order to verify their choice of object.

The concept of marking ahead is similar to the concept of accelerator keys in many of today's applications. A user is reminded of the keystrokes associated with particular menu items every time a menu is displayed since the name of the corresponding keys may appear next to the commands. The difference is that with our marking/pie menu mechanism, the user is not only reminded, but actually rehearses the physical movement involved in making the mark every time a selection from the menu is made. We believe that this further enhances the association between mark and command.

Another advantage of this mechanism is that it could be valuable for supporting fast performance on keyboardless computers. Many "pointer and keyboard" interfaces make extensive use of accelerator keys to speed menu selections. However, without a keyboard, an expert user is limited to making pointer-driven menu selections. If a pointing device like a stylus is used, our marking mechanism could replace the role of accelerator keys. A small set of short, straight marks could be associated with the most frequently used commands that do not otherwise have any obvious mnemonic marks to associate with them.

There are three advantages in associating short, straight marks for frequently used commands. First, the frequent use of the mark/menu will reinforce the association between mark and command: some marks can be remembered because they are mnemonic but short straight marks only can be remembered if they are used often. Second, reducing the articulation time of frequently used commands will produce more overall time savings than reducing the articulation time of rarely used commands. Third, computer recognition of straight marks can be very reliable and fast.

[1] Wait time is the cue to tell the system that no mark is about to be made. The menu is not popped up immediately for a number of reasons, including the fact that it can be distracting, can obliterate part of the screen, and takes time. Thus, especially for expert users, not requiring the menu to pop up supports the concept of a marking interface with none of the negative side effects of the presence of the menu. We have found in practice 1/3 second wait was not annoying for a novice and was long enough for an expert to avoid accidental menu pop-up.

Clearly there are several limitations to this technique. First, the marks made self-revealing by this technique are only those that are derived from the selection paths in a pie menu: straight strokes at various angles. Second, there are limits on the number of items that can be selected using this technique. An experiment has already been performed to reveal where performance begins to break down (Kurtenbach, Sellen & Buxton, 1991). We measured selection time and errors for menus containing from 4 to 12 slices, and also examined the effects of hiding the menus and requiring subjects to "mark ahead" rather than to select from visible menus. We found that subjects very quickly made the associations between marks and menu items. Further, for menus containing up to 5 slices, time to select and frequency of error using markings was equivalent to selection from visible pie menus. In addition, an odd number of menu items produced slower and more error-prone performance than an even number of items. But given that an even number of slices is used, as many as 12 slices per menu was shown to produce acceptable performance — a result we find most encouraging. These results indicate that by careful design of menu this technique can be used for selection of up to 12 items.

2.3 Operating on Individual Objects

Moving, copying, and deleting individual objects is also supported in GEdit.

Moving an individual object. Single objects are moved in GEdit by dragging. The technique used is similar to moving an icon in most direct manipulation systems: one points at the object to be moved, depresses the stylus or mouse button, drags the object to the desired position, then anchors it in that position by releasing.

Copying an individual object. There is no mechanism for copying individual objects. The rationale for this is that in this toy application, it is easier to create a new object than it is to copy an existing one.

Deleting an object. Another shorthand symbol can be used to delete individual objects. This is done by drawing through the object with a single horizontal stroke, as illustrated in Figure 3. The direction of the stroke does not matter.

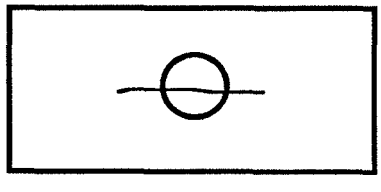


Figure 3. Deleting an individual object

Using a marking for deletion has similar advantages to those described for adding objects: Unlike direct manipulation there is no need for a delete mode or deletion tool. Also the deletion operation and object to delete can be expressed in the same action. However, there is no mechanism for making this mark self-revealing.

2.4 Operating on Groups of Objects

GEdit permits operations to be performed upon objects individually or in groups. The process of specifying the object(s) to be operated upon is known as specifying the "scope" of the operator. When the scope is a single object, we refer to it as "immediate" scope. All of the examples thus far have been immediate scope.

When the scope includes more than one object, we refer to it as "collective" scope. In GEdit, the mechanism for specifying collective scope is circling. Scoping mechanisms have a large impact on the effectiveness of a system (Buxton, Patel, Reeves & Baecker, 1981). GEdit provides an opportunity to explore some relatively unexplored aspects of scope specification, illustrated in the examples that follow.

Deleting a Group of Objects. A group of objects can be deleted by circling them, and extending the encircling line so that it terminates within the circle. This is illustrated in Figure 4.

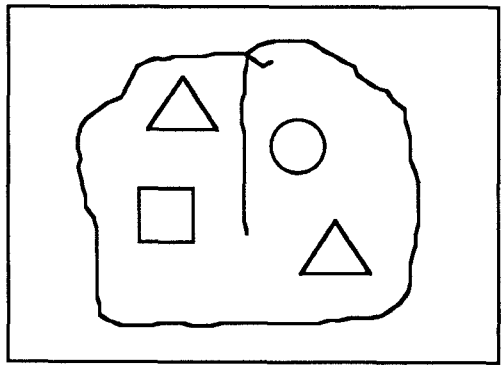


Figure 4. Deleting a group of objects

Moving a Group of Objects. The commonly used marking for "move" is used to move a group of objects. The objects to be moved are encircled, and the encircling line extended to the point to which the objects are to be moved (Figure 5).

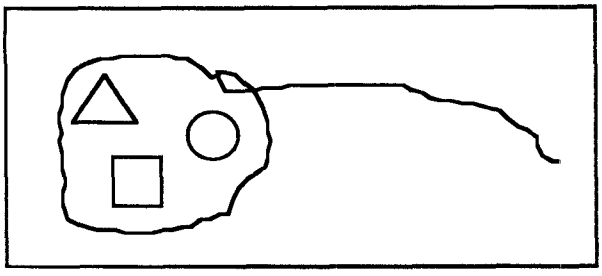


Figure 5. Moving a group of objects

Copying a group of objects. The command for copying groups of objects is similar to the move command. To copy, one adds a small "C" symbol to the end of the marking (Figure 6).

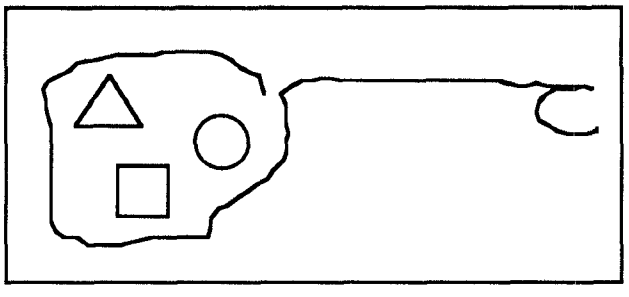


Figure 6. Copying a group of objects

The move command confronts us with two important points:

(1) Conflict between symbols: Note that in certain cases, there may be ambiguity between the move and the delete symbols. For example, a conflict could occur when the amount of movement is slight, resulting in the endpoint of the marking falling within the circle. (The endpoint being within the circle would be interpreted by the system as a delete command.) The example illustrates the fact that markings have impact on one another and must be considered in the larger context of an application.

(2) Lack of dragging: Unlike moving individual objects (immediate scope), groups of objects are not moved by dragging. When one moves objects by dragging, it is possible to view the placement of the objects before committing to that placement (releasing the mouse button). The inability to to *preview* the results of a group move or copy may or may not be a problem depending on the task. When working with graphical objects, this is may be a problem, since it hampers the precise placement of objects. On the other hand, it is not a problem if precise placement is not needed as in moving a group of icons to the trash can. The same is true for moving running text in a word processor. In this case, one doesn't want the indicated text to be moved until the insertion point is indicated.

Dragging groups of objects. The initial version of GEdit did not permit dragging of groups of objects. In order to support dragging, rather than introducing a new command such as"drag group of objects", we extended the "circle and move" technique. This was accomplished by combining marking and dragging in a single interaction. A similar technique has been described in Rubine (1991).

The collective move and copy commands are an extension of what was described above: after a user draws the circle and pointing tail, if pressure is maintained on the stylus and the pointer is kept still for a short period (approximately 1 sec), the objects appear to "slide" down the pointing tail and "lock" onto the cursor. The user can now fine tune the placement of the objects by dragging them around. This allows the user to preview their final position before releasing pressure on the stylus, committing to the final position of the objects (Figure 7). A move without a drag, as described earlier, can be done by immediately releasing the stylus at the end of the pointing tail. In this case the objects are immediately moved and coarsely placed.

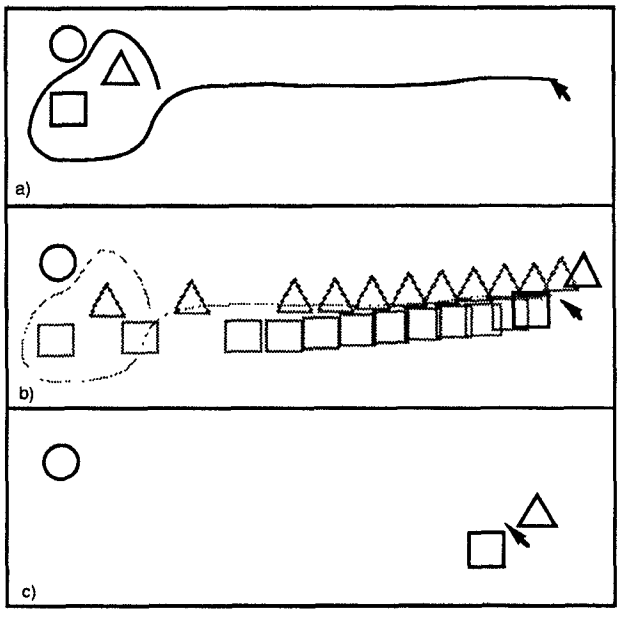


Figure 7. Moving a group of objects and then fine tuning their position by dragging. In a), the objects to be moved are circled and their destination pointed to. In b), the user waits with the stylus depressed and the objects "slide" to the approximate location. In c), the objects are dragged to an exact position by the user.

The same interaction can also be used for fine tuning the position of objects as opposed to moving them a large distance. One circles the objects to be positioned, and then keeps the stylus depressed and still. The objects then

become attached to the cursor and can be precisely positioned by dragging until the stylus is released. If the stylus is immediately released after drawing the circle, a collective scope group would be formed as described earlier.

The copy command has been extended similarly. If a user waits with the stylus depressed at the end of the "C", copies of the objects move down the pointing tail and their position can be fine tuned. If the user chooses not to wait, the copies are immediately moved and coarsely placed.

The interesting points concerning this group/position tuning technique are:

• We have used the event of keeping the stylus depressed and still as a signal to the system that the user wants fine control. This seems to work nicely as it is consistent with the user's need to focus attention on the task of precisely placing the objects.

• Having the system "slide" the objects to the new location provides essential visual feedback to indicate to the user when one can begin dragging.

• These interactions, (as in all interactions in GEdit), have been designed around the notion that a command and all its parameters are expressed a single, short and continuous moment — the stylus is depressed, then the command is articulated and then the stylus is released. This approach is based on Buxton's (1986) notion of "chunking and phrasing" where continuity of motion and physical tension are used to shape the structure of the dialogue in human-computer interactions. There is evidence that this type of design reduces user errors and increases performance (Sellen, Kurtenbach & Buxton, 1990).

2.5 Scoping

Deferred operation. The move and copy symbols share the property that the entire command (verb, direct object and indirect object) can be articulated using a single uninterrupted mark. Sometimes, however, this is not desirable. GEdit provides an alternative mechanism, whereby the scope and operation can be specified in two steps.

Scope can be specified without specifying an operator. The scope is first specified using a circling gesture, as illustrated in Figure 8. In this case, where no operator accompanies the specification of the scope, feedback is provided by way of a small box appearing at the end of the encircling line. This box is like a "handle" on the scope. Having thus specified the scope, one can then invoke the delete, move or copy operator by drawing the command part of the gesture, starting within this "handle."

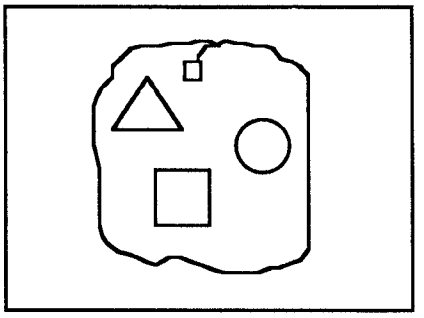


Figure 8. Specifying scope alone.

Scope with exceptions. One of the shortcomings of many scope specification techniques, such as circling or "drag through," is that they are all-encompassing. How does one handle cases where one doesn't want to operate on all of the encircled objects?

The main reason that GEdit includes independent specification of scope is to provide a means whereby some objects within the specified region can be excluded from being operated upon. This is made possible by allowing circles of exclusion as well as inclusion. This is illustrated in Figure 9. The outermost circle always indicates inclusion. Circles within circles toggle between exclusion and inclusion. In the example shown in Figure 9, all objects are selected except for the triangles.

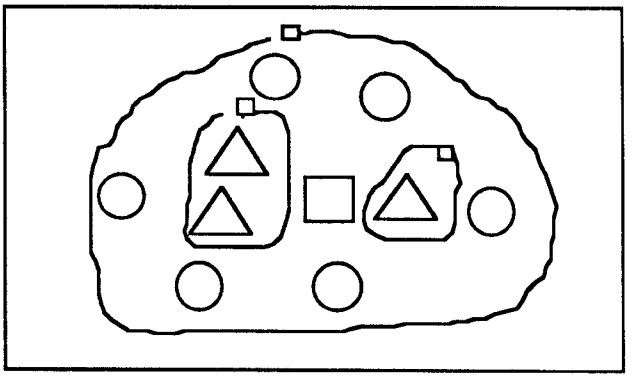


Figure 9. Specifying non-inclusive scope.

This technique is useful in cases where the objects to be operated upon are close to each other, but where there are other objects also in the same area. In the example illustrated in Figure 9, the intention is to operate on all objects except for the triangles. Deleting the intended objects could, for example, be accomplished by drawing a stroke starting from the outermost circle's "handle" and terminating anywhere within.

Once a mechanism is provided to specify scope incrementally, a logical extension is to enable disjoint objects to be selected. An example of this is illustrated in Figure 10. In this example, all of the triangles are selected through the use of two disjoint circles. Having come this far, however, we are confronted with a design dilemma.

We have established a convention that the square box at the endpoint of the scope-defining line is a "handle" from which any subsequent command must be initiated if it is to affect the encircled objects. However, in this case, there are two such "outer circles."

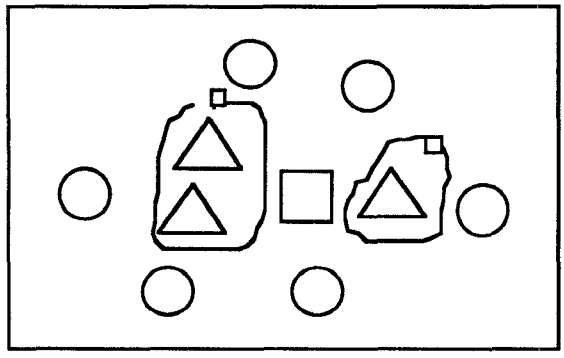


Figure 10. Disjoint scope specification.

The dilemma is, if both are defined, should an operation on one simultaneously affect the other? In GEdit, the answer is no—there is no notion of a single current selection set. Let us assume that the operation to be performed is "move". Using Figure 10 as an example, we would first move the triangle(s) in one circle, and then the other. While acting on one set of objects, the other set is unaffected until acted upon.

The disadvantage of this approach is that it takes two steps to do what might otherwise be done in one. It is also difficult to preserve the spatial relationship among the two sets of triangles during the move if this operation is done in two steps.

However this approach also has an advantage. In direct manipulation interfaces, typically there can only be one current set of selected objects (although this is generally an design choice not an inherent limitation of the direct manipulation paradym). If a user selects another set of objects the current selection of objects is lost. Often the user may not want to lose the original grouping of objects. Since GEdit has no notion of current selection, groupings of objects can be maintained while other objects are selected.

3. CONCLUSIONS AND FUTURE RESEARCH

We have described GEdit, a program used as a test bed for investigating features of markings and direct manipulation types of interfaces. This simple program has been very effective in exploring new interaction techniques. It has also been a useful vehicle for revealing design issues and has served as a tangible object for discussion.

This paper has demonstrated several ideas:

• Markings and direct manipulation techniques can complement each other within the same interface. However there are sometimes trade-offs between the two techniques.

• Markings and direct manipulation can be combined together within the same *interaction*. We have designed an interface which integrates marking and direct manipulation techniques to create commands where multiple parameters can be easily expressed within a single action.

• GEdit shows how one can exploit proof-reader and shorthand markings in way in which we hope reduces learning time (most people are somewhat familiar with proof-reader's marks). This technique also promises to be more efficient since it requires fewer actions for most commands and reduces the time it takes to execute these actions (i.e., shorthand symbols can be performed very quickly).

• GEdit has also shown how pie menus can be used in making shorthand marks self-revealing, thereby overcoming the problem of marks which are non-mnemonic.

There are three issues we hope to pursue in future research:

First, it would be interesting to evaluate the effectiveness of the self-revealing pie menu/marking technique in a real application. We are developing an HyperCard XCMD and X11 versions of the pie menu/marking software so application developers can use the technique. The software will also log when and where users use marks instead of waiting for menu pop-up. We hope this data will give us further insight into the effectiveness of this technique.

Second, we would like to explore ways of making other markings in GEdit self-revealing. One approach would be to supply an on-line graphical catalog of markings. However it is questionable how effective this would be in conveying information on more complex interactions such as the move/dragging mechanism. One approach we hope to explore is the use of animation to convey this more effectively.

Third, interactions which allow a user to make the transition from markings to direct manipulation dragging seem to be particularly powerful. It would be interesting to explore other applications where this could effectively be used.

Acknowledgements

The members of the Input Research Group at the University of Toronto provided the forum for the design and execution of this project. We especially thank Abigail Sellen for comments and assistance on this paper. We also gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada, Digital Equipment Corporation, Xerox PARC and Apple Computers.

REFERENCES

Buxton, W. (1982). An informal study of selection-positioning tasks. *Proceedings of Graphics Interface '82*, 8th Conference of the Canadian Man-Computer Communications Society, Toronto, May, pp. 323-328.

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

Buxton, W., Fiume, E., Hill, R., Lee, A. & Woo, C. (1983). Continuous hand-gesture driven input. *Proceedings of Graphics Interface '83*, pp. 191-195, Edmonton, May.

Buxton, W., Patel, S., Reeves, W., & Baecker, R. (1981). Scope in interactive score editors. *Computer Music Journal, 5 (3),* pp. 50-56.

Carr, R.M. (1991). The point of the pen. *Byte, Vol. 16(2),* pp. 211-221.

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

Hardock, G. (1991). Design issues for line driven text editing/annotation systems. Proceedings of Graphics Interface '91, Calgary, June.

Hutchins, E., Hollan J. & Norman, D. (1986). Direct manipulation interfaces. In D. Norman & S. Draper (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction,* pp. 87-124, Hillsdale, NJ: Erlbaum.

Kurtenbach, G. & Buxton, W. (1991). GEdit: A test bed for editing by contiguous gestures. *SIGCHI Bulletin.* pp. 22-26.

Kurtenbach, G., Sellen, A.J. & Buxton, (1991). Markings and pie menus: making stylus-driven interfaces self-revealing. submitted to *The Journal of Human-Computer Interaction* for publication.

Leitch, C. (1990). High-tech pen: Big deal or bust. *Toronto Globe and Mail,* Report on Business, pp. C1 & C11, Oct. 23.

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

Rebello, K. (1990). New PCs can kiss keyboards goodbye. *USA Today,* p. 6B,Feb. 22.

Rubine, D. H. (1990). *The automatic recognition of gestures.* Unpublished doctoral thesis, Dept of Computer Science, Carnegie Mellon University.

Sellen, A.J., Kurtenbach, G.P., and Buxton, W. (1990). The role of visual and kinesthetic feedback in the prevention of mode errors. *Proceedings of the Third IFIP Conference on Human-Computer Interaction, (INTERACT),* Cambridge, England.

Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology, 1,* 237-256.

Wolf, C.G. & Rhyne, J.R. (1987) A taxonomic approach to understanding direct manipulation. *Proceedings of the Human Facter Society—31st Annual Meeting,.* pp. 576-580

Wolf, C.G., Rhyne, J.R. & Ellozy, H.A. (1989). The paper-like interface, In G. Salvendy & M.J. Smith (Eds.), *Designing and Using Human Computer Interfaces and Knowledge-Based Systems,* Amsterdam: Elsevier, pp. 494-501.