GEDIT: A TEST BED FOR EDITING BY CONTIGUOUS GESTURES

GORDON KURTENBACH
BILL BUXTON

INTRODUCTION

GEdit is a prototype graphical editor that permits you to create and manipulate three simple types of objects using shorthand and proof-reader's type gestures. The three types of objects that can be manipulated are: squares, circles and triangles. Using hand-drawn symbols, the user can *add, delete, move* and *copy* these objects. Objects can be manipulated individually or in groups. Groups are specified with hand-drawn circling symbols.

GEdit provides a toy world that serves to demonstrate a number of concepts:

- The use of proof-reader's like symbols (rather than the more common alphanumeric character recognition);
- The use of *where* a symbol is drawn as well as *what* symbol is drawn to determine intent;
- The use of compound symbols, that is, symbols (such as move) that have more than one "token" embedded in a single continuous line. For example, in contrast to a character recognizer where what is recognized is at the level of the letter "A," the proof-reader's symbol 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 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. The hope is that it will provide a useful educational tool for those interested in character and gesture recognition. The intent of this document is to introduce the program and to serve as documentation for those using it.[1]

THE TOY WORLD

GEdit permits the creation and manipulation of three types of objects (squares, circles and triangles) in the context of a graphical layout program. It builds upon two earlier studies that investigated using similar symbols and objects (Buxton, 1982, and Buxton, Fiume, Hill, Lee & Woo, 1983).

This particular toy world was chosen because it is simple enough to be tractable, yet complex enough to capture a critical mass of real-world problems (such as one encounters in a CAD or graphical layout program).

BASIC OPERATION OF GEDIT

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

[1] GEdit is available from the authors at no charge. To receive a copy of the Macintosh version, just send the authors a blank formatted floppy disk and a mailing label containing your return address. Contact the authors for more information if you are interested in obtaining the Sun version.

Adding Objects

Objects are created using a form of shorthand notation. Each of the three objects is created using a specific symbol. These are illustrated in Figure 1:

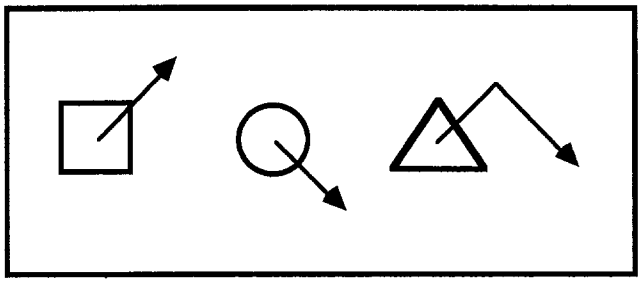

Figure 1: Shorthand Symbols for Adding Objects
Three objects can be defined: a square, circle and triangle. These are illustrated using bold lines. Shorthand symbols are used to define which type of object is to be created, and where it is to be placed. These are illustrated using thin lines with arrowheads. Object type is defined by the shape of the shorthand symbol. The new object is centred on the starting point of the defining symbol.

The object being defined is centered on the starting point from which the shorthand symbol is drawn. Thus, the symbol defines both *shape* and *position* of the new object.

Deleting an Individual 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 2.

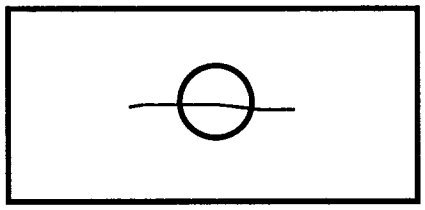

Figure 2: Deleting an Individual Object
Individual objects can be deleted by drawing a single horizontal stroke through them. The direction of the stroke does not matter.

The direction of the stroke does not matter.

Moving an Individual Object

Single objects are moved in GEdit by dragging. The technique used is the same as how icons are moved in most direct manipulation systems. For example, using a mouse as the pointing device, point at the object to be moved, depress the mouse button, drag the object to the desired position by moving the mouse, then anchor it in that position by releasing the mouse button.

Copying Individual Objects

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

SCOPE: IMMEDIATE AND COLLECTIVE

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 not so common aspects of scope specification. These are illustrated in the examples that follow.

COLLECTIVE OPERATIONS IN GEDIT

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 3.

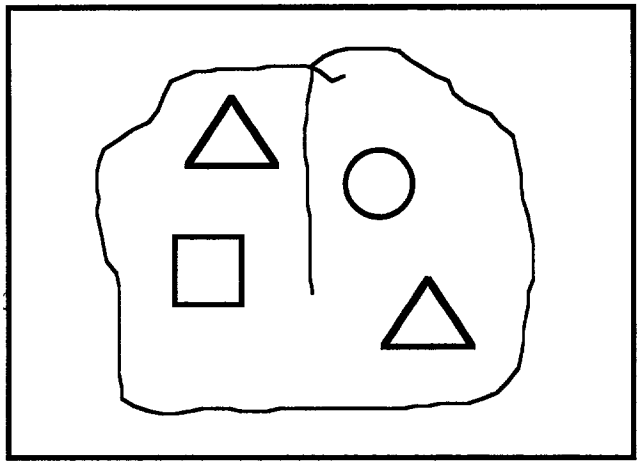

Figure 3: Deleting a Group of Objects
A group of objects is deleted by circling them, and extending the encircling line within the circle.

Moving a Group of Objects

The proof-reader's symbol 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. This is illustrated in Figure 4.

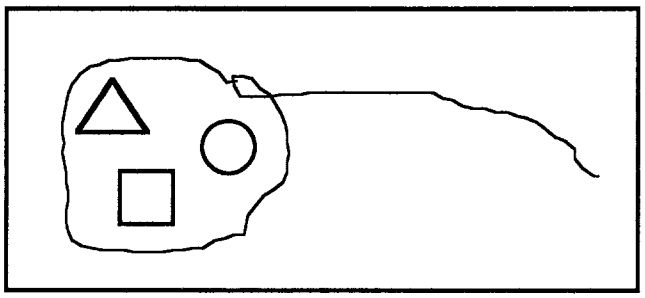

Figure 4: Moving a Group of Objects
A group of objects is moved by circling them, and extending the encircling line to the new location.

The move command confronts us with two important points worthy of discussion.

- *Conflict between symbols:* Note that in certain cases, there is a conflict between the move and the delete symbols. In particular, a conflict occurs when the amount of movement is slight, and would result in the endpoint of the defining line falling within the circle. The example illustrates the fact that gestures have impact on one another and must be considered in the larger context of an application.

- *Lack of dragging:* Unlike moving individual objects (immediate scope), the group of objects is not dragged to the new position., As a result, there is no possibility to preview the results of the operation before commiting to the move. When working with graphical objects, this is a problem, since it hampers the precise placement of moved objects. Note however, that if what was being moved was running text in a word processor, this would not be a problem. In this case, we don't want the indicated text to be moved until the insertion point is indicated. The example illustrates how the feedback of an action must depend on the type of data being acted upon as well as what operator is being used.

Copying a Group of Objects

The command for copying groups of objects is similar to that move operator. To copy, one adds a small "C" symbol to the end of the gesture. This is illustrated in Figure 5. The gesture works, but suffers from the same problems as the move gesture, namely, conflict with the delete command and lack of dragging, or preview. The issue of dragging is interesting, in that even if it were implemented, this particular approach to specifying "copy" would have

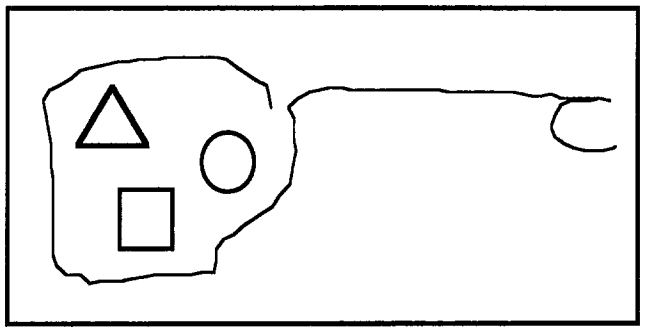

Figure 5: Copying a Group of Objects
A group of objects is copied by circling them, and extending the encircling line to the new location. and drawing a "C" (for copy) gesture.

a problem. The reason is that there is that the dragged objects have a "weight" that would make the execution of the "C" gesture inappropriate while dragging.

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 line. Sometimes, however, this is not desired. GEdit provides an alternative mechanism, whereby the scope and operation can be specified in two steps.

In such cases, the scope is first specified using a circling gesture, as illustrated in Figure 6.

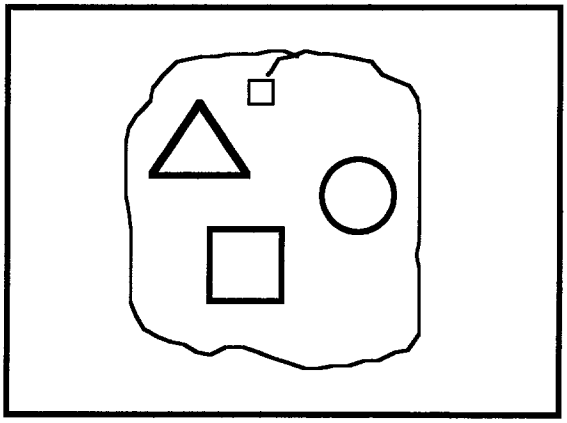

Figure 6: Specifying Scope Alone.
Scope can be specified without specifying an operator. In this case, by way of feed-back, the system provides a graphical "handle" in the way of a small square at the end-point of the circle. The delete, move, or copy gestures can then be articulated as long as the command portion is started in the box handle.

In this case, where no command 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."

Scope with Exceptions

One of the prime 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 prime reason that GEdit included the independent specification of scope, as discussed in the previous section, was to provide a means whereby some objects within the specified region could be excluded from being operated upon.

This is made possible by enabling circles of exclusion as well as inclusion. This is illustrated in Figure 7.

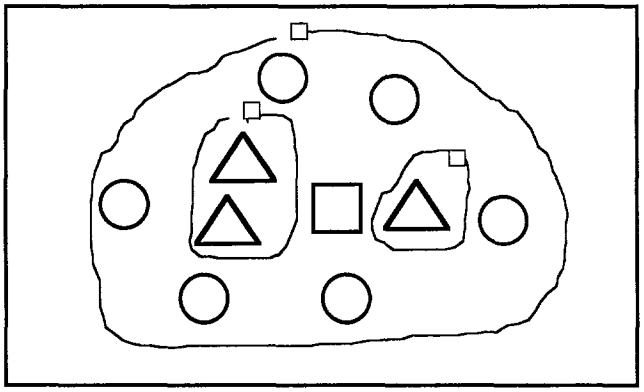

Figure 7: Specifying noninclusive scope.
Circles of both inclusion and exclusion can be specified. The outermost circle is always inclusion. Circles within circles toggle between exclusion and inclusion. In the example, all objects are selected except for the triangles.

This technique is useful in cases where the objects to be operated have a high degree of spatial coherence, but where there are other objects also in the same area. In the example illustrated in Figure 7, for example, 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.

DISCUSSION

GEdit is an example of what could be called a "line drive" interface. That is, an interface in which lines are used to articulate the user's intent. While GEdit is just a toy program, it brings forward a number of significant points. Some of these have been discussed above. A few additional points are discussed briefly in the remainder of this section.

Choice of symbol: Unlike menu-based systems, this class of interface is not *self-revealing*. It is not always obvious what commands are possible or how to invoke them. As the repertoire of commands grows, so does the problem of remembering the symbols that invoke them. Graphical mnemonics can help the situation through the use of symbols that have some graphical correspondence with what they represent.

In GEdit, for example, this is the case with the triangle symbol. On the other hand, this is not the case with the symbols for the square and circle, which were chosen for the speed with which they could be input. The program confronts us with the potential conflict between optimizing two aspects of human performance: memory and execution time.

Input device: The majority of studies of input technologies have focused on a limited number of tasks, especially pursuit tracking and target acquisition. These studies do not provide us with much information about how well various technologies an support the "line-drive" type of interface demonstrated in GEdit.

One advantage to the Macintosh version of the program is that it is very easy to change input devices for the purpose of comparison. Perhaps the most telling transaction is how consistently one can input the triangle symbol.

The recognizer is intentionally not very tolerant in what it will accept for this command. What users will quickly discover is that the mouse performs far worst than a stylus or the finger. This example makes the point that if the input device has been decided *a priori*, then the symbols must be designed to match the inherent strengths and weaknesses of that device. (Note, for example, that there is no problem articulating the delete, square or circle symbols with a mouse.)

Disjoint scope: 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 8.

In the 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."

The dilemma is, if both are defined, should an operation on one simultaneously affect the other? In GEdit, the answer is no. Let us assume that the operation to be performed is *move*. Using Figure 8 as an example, we would first move the triangle(s) in one circle, and then the other. The circle defining a particular group disappears after the operation on that group is finished. Other disjoint circles remain until acted upon.

While it works, this approach means that it takes two steps to do what might otherwise be done in one, and it is difficult to preserve the spatial relationship among the two sets of triangles during the move.

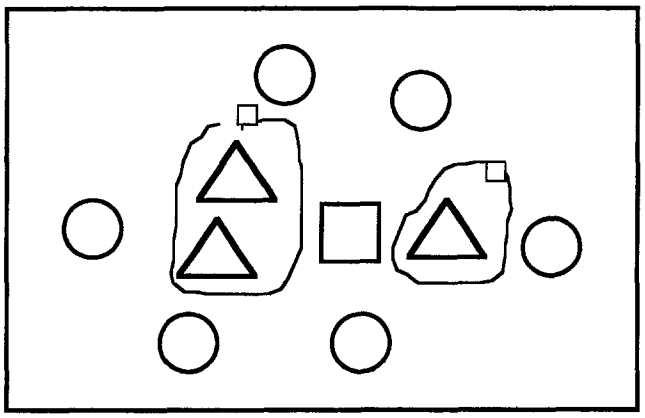

Figure 8: Disjoint scope specification.
Ideally, disjoint clusters of objects can be selected as the scope of an operator. In this case, multiple circles are used. In contrast to Figure 7, in this case, only the triangles are selected.

Temporal pervasiveness of scope: The previous example serves to illustrate a further point. With direct manipulation interfaces, there is no systematic way in which scope can remain in effect from operation to operation, especially if interspaced with any other transactions (such as window manager commands) not acting on the selected objects.

This is an unsolved problem. Without imposing a burden larger than the problem that we are trying to solve, how can we easily indicate when we want the scope to persist from transaction to transaction? Using disjoint scope specification, as in Fig. 8, scope persists until the objects are operated upon. This hints at persistence. However, the range of operations in the program is not rich enough to really push on the issue. For example, the issue would be far more germane if we could select a group of objects, move them, add a few more, then scale or rotate all of them.

The transaction is representative of a number of real-world tasks that are currently poorly supported. It is a shortcoming of GEdit that it is not rich enough to allow us to fully explore this issue. Nevertheless, even if it provides a hint that the issue needs attention, it has served some useful purpose.

CONCLUSIONS

GEdit is a seemingly simple program that affords us the ability to gain valuable experience in a relatively uncharted area of HCI. The value of GEdit is not so much in how it has implemented its solution, as in its bringing the issue to the fore. We hope that it makes some contribution to the general literacy about "line drive" interfaces, and helps demonstrate the value of such critical-mass toy programs.

ACKNOWLEDGEMENTS

The research described in this paper has been undertake with the generous support of the Natural Sciences and Engineering Research Council of Canada, Xerox PARC, Apple Computer and Digital Equipment Corp. This support of our laboratory and research is gratefully acknowledged.

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, 1982, 323-328.

Buxton, W., Fiume, E., Hill, R., Lee, A. & Woo, C. (1983). Continuous Hand-Gesture Driven Input. *Proceedings of Graphics Interface '83*, Edmonton, May 1983, 191-195.

Buxton, W., Patel, S., Reeves, W., & Baecker, R. (1981). Scope in Interactive Score Editors. *Computer Music Journal* 5 (3), 50-56.