# CONTINUOUS HAND-GESTURE DRIVEN INPUT

*William Buxton*
*Eugene Fiume*
*Ralph Hill*
*Alison Lee*
*Carson Woo*

Computer Systems Research Group
Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A4

*ABSTRACT*

We are interested in the use of physical gesture in dialogues between humans and computers. A case study of an implementation of a simple sketch editor is presented. In the current study we explore the use of sketching hand gestures to input a type of graphical "short-hand." The command interpreter is, consequently, a type of character recognizer.

The implementation is restricted to using a vector-drawing display for output, and a graphics tablet with a 4-button puck for input. The basic hypothesis was that, with appropriate feedback, the richness of tablet-based input could be further exploited. No formal experiments were run. Rather, a simple system is presented whose fluency of dialogue structure argues for the need of increased attention to such input techniques.

A sketch editor was chosen for the case study because it forced one important issue: that of self-similarity. The interesting thing about the system is that *everything* expressed by the user -- commands, scope, arguments, and sketch data -- is articulated using the same sketching techniques. To extend this point to the extreme by way of example, a sketched circle may invoke the command to delete all of the sketches of circles within the scope of an enclosing sketched circle. The system demonstrates that that which is confusing in words, can be effective and natural in gesture. Furthermore, it demonstrates that the problems of ambiguities (such as when is a circle a circle, command, or scope?) can be resolved without resorting to overly complex syntax.

KEYWORDS: graphical input and interaction, user interface design, tablet-driven input, character recognition, CAD, editing.

## 1. INTRODUCTION

In everyday life, physical gestures are a powerful means of communication. A set of physical gestures may constitute an entire language, as in sign languages. They can economically convey a rich set of facts and feelings. For example, waving one's hand from side to side can mean anything from a "happy goodbye" to "caution". Use of the full potential of physical gesture is also something that most human-computer dialogues lack.

The need for improved techniques of graphical input and interaction are becoming increasingly recognized [GIIT83]. In this study we explore the use of sketching gestures on a graphics tablet to input command statements in a type of "short-hand." Some effective tablet-based techniques have already been catalogued in Evans, Tanner, and Wein [EvTW81]. The concept of graphical shorthand interpreted by a character-recognizer has also been seen in the literature. Teitelman developed a trainable character recognizer as early as 1964 [Teit64]. Coleman developed a simple text editor that was driven by on-line hand-drawn proof-reader's symbols [Cole69]. In a musical score editor developed by Buxton *et al.* [BSRP79], hand-drawn curves were used to identify the notes to be used as the scope of an operator. Buxton also demonstrated how a simple shorthand

could be used to effectively select and position shapes in graphical layout [Buxt82].

A major objective of the current paper is to re-examine systems driven by continuous hand gestures, such as those cited above. Our purpose was to gain, through actual experience, a better understanding of how the potential benefits of such systems could be realized in a practical sense using contemporary technology. This interest was prompted by a perceived discrepancy between the apparent power of the approach and its extremely low utilization in current practice (such as in text editing and CAD). Our approach, therefore, took the form of a case study in which we implemented a simple sketch editor. We believe that this study demonstrates the viability of the approach using current technologies. Our experience with the system also leads us to conclude that for such systems to come into common practice, improved support tools must be provided to the applications programmer. Part of our presentation, therefore, discusses our recommendations concerning such tools.

The gesture-based commands found in this editor are common to virtually all editing and design environments. Since all gestures are made on a graphics tablet, we believe our study has implications for CAD systems, text editors and personal workstations. Some implications well-chosen gestures:

(1) reduce the apparent syntactic complexity of dialogues and the number of modes [Tesl81] occurring within a dialogue.

(2) enhance the performance and learning of tasks that are difficult to verbalise.

(3) increase a system's ease of operation and the user's ability to combine gestures into appropriate tasks.

(4) may decrease the number of input devices required, and may increase the utilisation of specific input devices.

(5) are likely to be difficult to implement using current software tools.

The first four implications suggest that overall human interaction is likely to be improved through the use of gesture. Indeed, this is implied by the keystroke model of Card, Moran, and Newell [CaMN80] when considered in combination with point (4). As will be seen, point (5) is a result of limitations in present day programming environments, which do not provide adequate tools for dealing with gesture recognition. An objective of this paper, therefore, is to use a demonstration of the power of the approach as a means to stimulate the development of such tools.

## 2. DESIGN ISSUES AND SELF-IMPOSED CONSTRAINTS

### 2.1. Environment

The environment in which the system was to run was decided at the outset. While this approach is inconsistent with a strict "levels of abstraction" design philosophy, it appears to be the only realistic way to assess the feasibility of operations and gestures. Also, it ensures that we would not design a system that could not be realistically implemented on the target environment, which consists of a PDP 11/45 computer running the UNIX[1] operating system, to which a fast line-drawing display and a graphics tablet, with a four-button puck, are attached.

### 2.2. Sketch Editor Design

The sketch editor permits users to draw simple free-hand curves, and supports the following operations on them:

(1) move part of a drawing to another location.

(2) copy part of a drawing to another location.

(3) save a drawing in a permanent store.

(4) restore a drawing from permanent store.

(5) quit.

### 2.3. Choice of Gestural Tokens

We began our work by cataloguing a list of observable puck motions that could conceivably be understood as gestures. The list included recognising shape, orientation, size, proportion, velocity, and timing. This gave us a great deal of flexibility in patterns for individual tokens. It was a flexibility that turned out to be essential, because the different puck motions can be recognised with varying degrees of accuracy on our system, as will be discussed shortly.

Our basic gesture design criterion was to choose gestures that are analogous to those used by someone doing pencil and paper sketches. Imagine a sheet of paper with various objects pencilled on it. To draw another object, you would simply draw it at the desired location on the paper. In a subsequent version of the drawing, you may decide that an object should be moved elsewhere on the page; in this case, you might draw a circle or other curve around the object and then sweep the pencil across to the new location of the object. If a copy is desired instead, you might

1. "UNIX" is a trademark of Bell Laboratories.

similarly draw a curve around the object, make some mark on the paper that a copy is to be made (as opposed to a simple move), and again sweep the pencil to the desired location. Notice that the gestures tend to be fluid and continuous. The task is to develop gestures that closely resemble these, and which can be reliably recognised by a simple program.

To demonstrate just how under-utilised input devices often are, and to minimise the difficulties in locating buttons on our (poorly designed) puck, we decided to restrict ourselves to implementing all gestures using a single button of the puck, together with the graphics tablet. This decision also leaves the user free to use a single button stylus, which may be a more natural device for sketching and gesturing. We have developed simple gestures that can be made with these devices, which can be used for saving, restoring and quitting; however, for this discussion, we will concentrate on the four basic operations: draw, move, copy and delete[2].

The commands implemented follow an infix format:

<scope> <operation> <target>

where the scope defines the domain of the operator. The target field is used in the *copy* and *move* commands, where the new location must be specified. In commands that require all three tokens to be specified (move and copy), it is interesting to note how all three flow together into a smooth gesture. This is important since some designers avoid syntax in which there is more than one argument to an operator. This is evident in the XEROX Star, for example, which requires three distinct actions (copy, delete, copy) to carry out a simple move command [SIKV82].

*Drawing* is indicated simply by doing it: move the puck to where you wish to begin, depress the button, and draw; drawing terminates when the button is released.

The *move* and *copy* commands are more complicated, as it is necessary to specify the scope of the command, the operator, and the target. This is done by encircling the desired curves, indicating the operation, and then moving the puck (thereby dragging the indicated curves to the desired location). This is illustrated for the *move* command in Figure 1. Thus, an entire move or copy operation is performed in one continuous gesture.

2. A videotape of our system is available which demonstrates the basic gestures and the general use of the system. See [BFHL83].

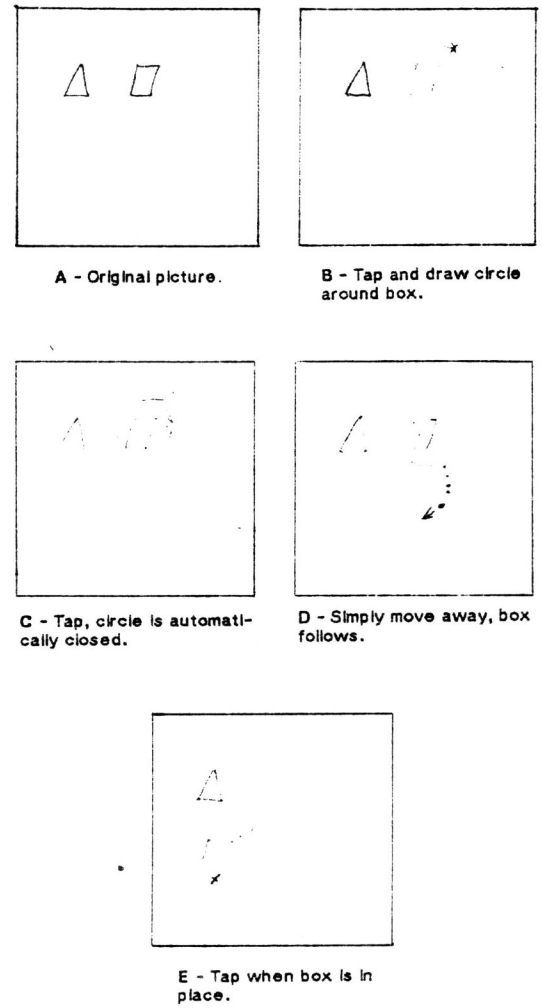


Figure 1: MOVE Command Sequence

*SCOPE:* The beginning of a scoping gesture is indicated by a timing cue: a quick tap of the puck button. The tracking symbol on the display then changes to an icon connoting the expectation of a scoping gesture. The user then either holds down the button to draw the scope and subsequently releases the button to indicate scope completion, or, alternatively, draws the scope with the button up and subsequently taps the button to indicate scope completion. The system draws a straight line from the last point of the scope gesture to the starting point, to close the scope, if necessary. There is no restriction on the complexity of the scope that may be drawn, although the meaning of "inside" can become unintuitive for convoluted

scopes. On scope completion, the tracking symbol is changed to an icon which connotes the expectation of a command gesture (which may be cancelled by the user, by tapping once again).

*COMMANDS:* A move command is gestured simply by moving to the desired location after completing the scope. A copy command is indicated by making a "C" like gesture and then moving to the desired location. Objects are *deleted* from the drawing by moving them off the drawing area. While the user is making the command gesture, the contents of the scope is calculated and this object now becomes the tracking symbol. A final tap of the button anchors the object when the desired location is reached.

We would like to stress two points about our design. First, the complexity of a gesture is inversely proportional to its expected frequency of use. Thus, drawing is assumed to be the most common command, followed by moving and copying. Second, the gesture for a copy command is an obvious, and easily remembered symbol. The gestures for save, restore, and quit also involved the use of easily-drawn and remembered characters. Auxiliary use of the keyboard or menu selection is not required[3].

### 3. IMPLEMENTATION DIFFICULTIES

The above discussion gives only part of the story regarding the design and choice of gestures in our sketch editor. As we suggested earlier, a variety of possible puck motions can be recognised as gestures. Unfortunately, all cannot be recognised with equal fidelity in a time-sharing environment. In particular, velocity-sensitive gestures and timing cues can be difficult to interpret precisely.

Recognition problems result from the need for very fine grained temporal analysis. The timing information at the level of granularity necessary for this analysis is not available in a timesharing environment.

It can also be extremely difficult to quickly recognise complex shapes, such as arbitrary characters. This has been an active research problem in artificial intelligence and image processing for several years. Unless one wishes to spend many human-months building an extensive character recogniser, one must be very modest in the selection of command gestures[4]. The point we wish to make is that gesture recognition can be difficult both because of environmental restrictions, and because the necessary gesture recognition tools simply do not exist. Because of these restrictions, and the lack of adequate tools, our system, like most present-day interactive applications, had to be built from scratch. It thus required greater implementation (and design) effort than should have been necessary.

We found the lack of communicating, concurrent processes in the UNIX environment to be a surmountable but significant drawback. Since scope determination requires considerable computation time, it should be run in parallel with the user's selection of a move or copy command. Currently, we compute scope contents after the command has been understood (which is usually *before* the user thinks it has been understood). Fortunately, for scopes of moderate complexity, scope contents determination does not create an annoying delay, as the user usually starts positioning the (expected) contents as soon as the command gesture is given. During this time, the tracking symbol is a Buddha, indicating that the contents will appear shortly, and encouraging the user to be patient. This simple concurrency (a symbol tracking the pointing device independent of other processing), is provided by our graphics package, GPAC [Reev80], and is a major factor contributing to the usability of the system.

In spite of our difficulties, we believe we have demonstrated both the utility of gestures and the need for good gesture recognition tools. Ultimately, such a facility should be an integral part of a programming environment. There are at least two ways in which this can be accomplished. One approach involves integrating a catalogue of gestures into a powerful User Interface Management System (e.g. FLAIR [WoRe82], and Menulay [BLSS83]). The extensibility of such an approach must be carefully considered. Indeed, the impact of gesture-driven input on UIMS's remains to be assessed. An alternative approach is to incorporate gesture-recognition tools into object-based programming environments such as Smalltalk [Inga78] and ThingLab [Born81]. This is advocated in [Fium83]. While this proposal likely to be more flexible than the first, it is equally likely that it will be more resource-intensive. In any case, an implementation of either approach will be a significant improvement on the current situation.

We believe that many timing difficulties would be eliminated if our system were running on a stand-alone

3. Clearly a keyboard-like device is required to enter new file names. Better schemes exist for selecting existing files. See [BSRP79], for example.

4. In fact, recognition of the character "C" is easy: three invisible axes must be crossed in the correct order, starting from the upper right.

system with a reasonably fast processor. Thus gesture-based text-editors, VLSI-layout systems, and other interactive applications on personal computers are certainly possible, and indeed desirable.

## 4. CONCLUSIONS

The need for more "natural" interaction techniques has been often stressed. This paper makes the modest suggestion that gesture-based input is such a technique. We have demonstrated its effectiveness with dialogues that are common to many interactive applications. Research into the uses of gesture in human-computer interaction is embryonic, and we hope to have inspired others to exercise their ingenuity in developing effective gestures.

Experimentation with gestures and their composition is essential. For this, powerful programming environments are required. These environments do not currently exist, and consequently gesture-driven dialogues can be unreasonably difficult to implement. Future programming environments must facilitate experimentation with gesture-based dialogues with the same simplicity and efficiency that parser-generators currently permit with command languages, and User Interface Management Systems permit with menus.

## 5. REFERENCES


**BFHL83** Buxton, W., Fiume, E., Hill, R., Lee, A., and Woo, C., "Etch: A system based on continuous hand-gesture driven input", videotape, 1983. Write to: W. Buxton, Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 1A4.

**BLSS83** Buxton, W., Lamb, M.R., Sherman, D., and Smith, K.C., "Towards a comprehensive user interface management system", extended abstract elsewhere in this issue, full paper to appear in *Computer Graphics 17*, 3, (July 83).

**Born81** Borning, A., "The programming language aspects of ThingLab, a constraint-oriented simulation laboratory", *Transactions on Programming Languages and Systems 3*, 4 (Oct. 1981), pp. 353-405.

**BSRP79** Buxton, W., Sniderman, R., Reeves, W., Patel, S., and Baecker, R., "The Evoulution of the SSSP Score Editing Tools", *Computer Music Journal 3*, 4, pp. 14-25.

**Buxt82** Buxton, W., "An Informal Study of Selection Positioning Tasks", *Proceedings of Graphics Interface '82*, Toronto, May 1982, pp. 323-328.

**CaMN80** Card, S.K., Moran, T.P., and Newell, A., "The Keystroke-Level Model for user performance time with interactive systems", *Comm. ACM 23*, 7 (July 1980), pp. 369-409.

**Cole69** Coleman, M. L., "Text Editing on a Graphic Display Device Using Hand-Drawn Proofreader's Symbols", *Proceedings of the 2nd University of Illinois Conference on Computer Graphics*, 1969.

**EvTW81** Evans, K.B., Tanner, P.P., and Wein, M., "Tablet based valuators that provide one, two, or three degrees of freedom", *Computer Graphics 15*, 3, (Aug. 81), pp. 91-98.

**Fium83** Fiume, E.L., *A Programming Environment for Constructing Graphical User Interfaces: A Proposal*, M.Sc. Thesis, Department of Computer Science, University of Toronto, January 1983.

**GIIT83** "Graphical Input Interaction Technique (GIIT) Workshop Summary", *Computer Graphics 17*, 1 (Jan. 1983), pp. 5-30.

**Inga78** Ingalls, D.H.H., "The Smalltalk-76 programming system: design and implementation", *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, Jan. 23-25, 1978, pp. 9-16.

**Reev80** Reeves, W.T., *GPAC Users' Manual (Fourth Edition)*, Computer Systems Research Group, University of Toronto, 1980.

**SIKV82** Smith, D., Irby, C., Kimball, R., Verplank, B., and Harselm, E., "Designing the Star User Interface", *Byte 7*, 4 (April 1982), pp. 242-283.

**Teit64** Teitelman, W., "Real-Time Recognition of Hand-Drawn Characters", *Fall Joint Computer Conference 1964*, Spartan Books, Baltimore, Md., p. 559.

**Tesl81** Tesler, L., "The Smalltalk Environment", *Byte 6*, 8 (Aug. 1982), 90-147.

**WoRe82** Wong, P.C.S., and Reid, E.R., "FLAIR--User interface design tool", *Graphics Interface '82*, May 1982, pp. 15-22.