

Washington University in St. Louis

Washington University Open Scholarship

All Computer Science and Engineering
Research

Computer Science and Engineering

Report Number: WUCS-92-35

1992-08-01

Gesture System for a Graph Editor

Burak Muammer Taysi

This thesis investigates a process of designing a gesture system for a simple graphical editor on a pen-based computer. The graphical editor, named Box and Arrow Editor, was developed to test the designed gestures. The results of the testing and the evaluation of the principles used to formulate the gestures are presented. The design process covers issues of application of principles to gesture design, elimination of all menus, user evaluations and recognition problems.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Recommended Citation

Taysi, Burak Muammer, "Gesture System for a Graph Editor" Report Number: WUCS-92-35 (1992). *All Computer Science and Engineering Research*.
https://openscholarship.wustl.edu/cse_research/598

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

Gesture System for a Graph Editor

Burak M. Taysi

WUCS-92-35

August 1992

**Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899**

A thesis presented to the Sever Institute of Washington University in partial fulfillment of the requirements for the degree of Master of Science.

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

GESTURE SYSTEM FOR A GRAPH EDITOR

by

Burak M. Taysi

Prepared under the direction of Professor T. D. Kimura

A thesis presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

August, 1992

Saint Louis, Missouri

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

GESTURE SYSTEM FOR A GRAPH EDITOR

by Burak Muammer Taysi

ADVISOR: Professor T.D. Kimura

August, 1992

Saint Louis, Missouri

This thesis investigates a process of designing a gesture system for a simple graphical editor on a pen-based computer. The graphical editor, named Box and Arrow Editor, was developed to test the designed gestures. The results of the testing and the evaluation of the principles used to formulate the gestures are presented. The design process covers issues of application of principles to gesture design, elimination of all menus, user evaluations and recognition problems.

TABLE OF CONTENTS

CHAPTERS	Page
1. INTRODUCTION	1
2. BACKGROUND	5
3. BOX AND ARROW EDITOR	14
4. GESTURE SYSTEM FOR BAE	17
5. IMPLEMENTATION	42
6. EVALUATION	53
7. CONCLUSION	62
8. ACKNOWLEDGEMENTS	64
9. BIBLIOGRAPHY	65
10. VITA	71

LIST OF FIGURES

FIGURES	Page
1. The Move Gesture	2
2. Deleting in GEdit a) individual b) group	9
3. Different Gesture Sets used in GDP	10
4. Pen Windows Gestures	12
5. PenPoint Gestures	13
6. Sample Box-graph	15
7. BAE Commands and Gestures	20
8. Open Command Window	24
9. SaveAs Command Window	26
10. Close Command Window (with Save)	26
11. Close Command Window (without Save)	26
12. The Duplicate Command	28
13. Grid Command Window	31
14. Align Left Command	33
15. Align Horizontal Command	34
16. Lombard-NeXT Hardware Connection	43
17. Software Architecture	44
18. Screen Layout for BAE	47
19. Recognition Process	48
20. Sample Gesture (Paste)	48
21. Grid Features	50
22. Energy Features	50
23. Gestures Inputted by Kids	56
24. The 'box' gesture entered by a Child	56
25. The 'paste' gesture entered by a Child	57
26. The '2 Headed Arrow' (normal) gesture inputted by a Child	57

LIST OF TABLES

TABLES	Page
1. Description of Types	14
2. Draw Commands	45
3. Test Results	54

Short Title: Gesture System for Graph Editor Taysi, M.S. 1992

1. Introduction

The research on pen based computing began in the mid 1960's with the development of Rand and Sylvania tablets [1,2], and has seen significant development in the late 1980's into 1990's. Referred to as both the Paper-Like Interface [3]¹ and Silicon Paper Technology [4], the advantages and possible uses for this type of computing have been under investigation for some time. The pen computer brings with it a new way of interacting with the computer, namely handwriting and gestures. A gesture is a motion which conveys meaning; examples are conducting music, sign language and sports signals, such as the ones in baseball [44]. An example of a gesture system in a two dimensional form would be proof-readers marks which are used to convey commands to the writer about needed changes.

This research concentrates on the use of 'gestures' as inputs to computers (specifically pen computers) to represent a specific command. "'Gesture' refers to both the resulting visual form and the temporal characteristics of the drawing process" [9]. The growing interest in pen computers has brought the study of gestures to the forefront as an effective means to express commands to the computer.

Gestures have the power to specify the operand and the operation simultaneously with a single stroke, replacing the multiple step selection and execution of a command by the mouse in the window/menu based traditional user interface. A stroke is defined to be the pen input from pen down to pen up. Here the terms 'pen up' and 'pen down' signify the status of the pen with the tablet; if in contact then the pen is 'down'; otherwise the pen is 'up', and the pen input is the coordinates of the pen on the tablet when the pen is down.

The 'move' gesture for a graph editor, shown in Figure 1 [39], is a good example of this powerful interaction technique. With a single stroke the user is able to identify the object and then specify the new target location with the end point. When a point in a

¹The Paper-Like Interface is a trademark of IBM Corporation.

gesture, like the end point of the 'move' gesture, carries a special meaning, it is said to be a *hot point*. Hot points are one of the special features of gestures that makes it both a powerful user interface and difficult to recognize.

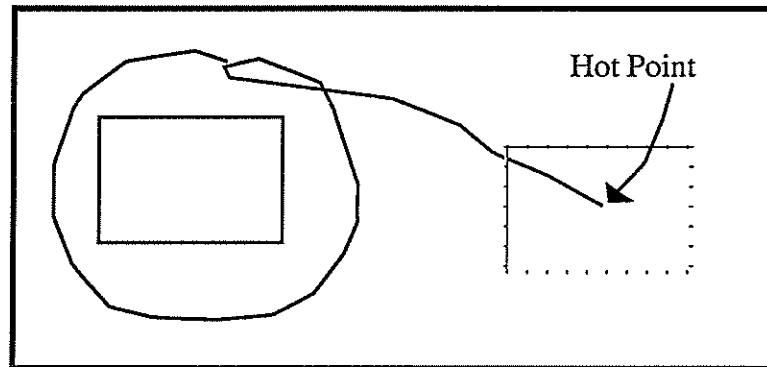


Figure 1. The Move Gesture

Any research that deals with communications between humans and computers fall under the category of User Interface research. Gestures are means of communication between humans and computer, and a study of gestures is a part of a new research area of what we call Pen User Interface. As the use of gestures becomes more common, we expect to see more complex gesture systems. With the increase in complexity, the design of such systems becomes more difficult. As a user interface tool, gestures should be easy to remember for humans, and easy to be recognized by computers. The designer needs to address both the needs of the user and the incorporation of recognizer's strengths into the gesture design.

This problem brings up some important research issues in dealing with gestures. One of them is formulating design principles for gesture systems. There are common sense guidelines that are used to generate gestures, like ease of use, rememberability and simplicity. However, there has not been any investigation into application of these principles to a pen computer gesture system. In this thesis we conducted this investigation. We show how each of these principles were applied in the gesture design phase, and report, in the evaluation section, on what changes needed to be made to

effectively use these principles. We report on the problems encountered and how they were dealt with. If an effective principle for designing gestures for pen computers is formulated, then it could be applied to many new pen computer applications. Two such applications are discussed below.

A specific area that will be effected with this new user interface will be the area of visual programming languages [36]. Since pen-based computers represent a new dimension in computer and user interaction, the visual languages that are designed for such systems have to be aware of the difference in the interface characteristics [35] of the pen-based computers. Visual languages for pen computers, such as Hyperflow [34], will be using gestures as the main tool of interaction between the programmer and the computer. Hyperflow is an extension of a visual programming language, Show and Tell [12], which is based on the mouse/windows user interface. As in Show and Tell, Hyperflow syntax will be based on the use of boxes and arrows, boxes representing processes, and arrows representing data flow between processes.

Another use of this method will be to obtain gestures as a user interface for children on a pen-based computer. Kumon Machine [45] is a prototype pen-based computer, that will be used to teach mathematics to children between pre-school and 12th grade. The interface between the Kumon machine and the children will involve the use of gestures, handwriting, drawing and keyboarding.

We conducted a preliminary investigation into designing gesture systems for pen-based applications, by implementing a simple graph editor, called Box and Arrow Editor, whose commands are represented by gestures instead of menu items. The investigation involved building a taxonomy from the existing gesture systems, a study of the gesture recognizer, designing of the gestures, and finally evaluating the designed gestures.

Chapter 2 provides a background for this area of research. It highlights some of the current gesture systems being used and tested. It also looks at some of the work that has been performed in the area of gesture recognition.

Chapter 3 introduces the Box and Arrow Editor (BAE). The purpose of the Editor is outlined, along with formal definitions of the objects and operations supported by BAE. Finally the problem of designing a gesture syntax for this system is stated.

Chapter 4 explains the requirement specifications for gesture design. The gesture syntax is then developed using these specifications.

Chapter 5 reports the implementation of BAE. It covers hardware and software architecture, a description of the gesture recognizer used, along with information on gesture data collection and training.

Chapter 6 evaluates the system. The results of the tests are given. It reports on how the gestures satisfied the requirements specifications for gestures. The results of the investigation are summarized. We also give an explanation of the recognizer's strengths and weaknesses in its recognition of gestures, and describe better techniques of reporting the average recognition rate for a better analysis of recognizers.

Finally Chapter 7 gives the conclusion including possible future work.

2. Background

2.1. Earlier Studies

With the latest surge of technical improvements in tablets and increased interest in pen-based computers, the research into the feasibility of gestures as an effective human computer interface has also increased. Earlier studies on gestures confirmed that there has been a high inter-user consistency with the use of gestures for a particular operation [5,6,7,8], especially in the area of editing. Other studies that dealt with direct manipulation, defined in [9], also saw the plausibility of gestures as a good technique for direct manipulation [10]. These and other research such as [11,12] showed the potential of gestures over keyboard and mouse as an effective interaction tool. The power of the gesture is more realized when dealing with tasks that use a combination of both text and graphics.

Earlier studies also established the terminology to describe gestures. The functional elements of gestures were stated by [13] as being *operations*, *scopes*, *targets*, *literals*, and *modifiers*. The act of selecting one or more objects to be manipulated is called *scoping*. *Target indication* simply refers to the specification of location of interest. *Operations* are the actions that the gestures perform. *Literals* and *modifiers* are handwriting used to add or replace text and used as parameters to operations, respectively. For example if a month is crossed out and the new month written over it, the writing is a literal, but if text is selected and the letter 'u' is written for upper casing the selection, then the 'u' is a modifier. The same paper also details how a single stroke of a gesture has many functional elements, and the difficulty this presents to the recognizers. The issues of contextual help and problems with determining *closure* were also brought up. *Closure* represents an event which signifies the end of a command. The return or enter key is an explicit closure event for many applications that use a keyboard interface. The problem with closure arises when there are multiple commands that are identical except that one has an additional final event, like an additional stroke.

2.2. Recognition

An important issue when dealing with gestures are their recognition. Character and gesture recognition need different approaches. Gestures use hot points, changing sizes and changing shapes to convey meaning. The approaches taken in character recognition, described in the next section, do not address these special issues effectively. Even though some principles of character recognition can be used to design a gesture recognizer, it has to be different from character recognizer. Some of the latest gesture recognizers are detailed later.

2.2.1. Character Recognition

Many studies have been made for character recognition. In handwriting, elastic matching has had good success [14,15,16]. Even with all the improvements in the tablet technology, studies confirm that surface difference and the parallax problem (the appearance of electronic ink away from the tip of the stylus) are still contributing to badly written inputs and causing problems for recognition systems [17,18]. These studies conclude that more hardware support is needed to improve the recognizers. Other studies improved on elastic matching, by using different techniques, such as recognize-then-segment recognition [19]. The recognize-then-segment recognition first classifies the strokes as they are inputted, generates character hypotheses, then uses these hypotheses to estimate the optimal character sequence for each word. Another technique is recognizers that decompose a character-match distance into the sum of stroke-match distances and stroke-relationship terms [20]. Character-match distance refers to the difference between the character and the prototype. Stroke-match distances refers to the difference in strokes in character and strokes in the prototypes. Stroke relationship refers to the ordering of the strokes in characters. For example, 'A' would be made up of three strokes ' / ' \ ' and ' - ', in that order. There have even been studies that help teach the users the weaknesses of the recognizers so that the users adopt their writing for better recognition[21].

In the application program MEADOW, which is designed to help with math problem solutions, new issues about pen-based computing were discovered when the system was tested with children ranging from 6 to 11 years old [24]. This study pointed out that with children stroke-based recognition algorithms were not reliable since children this young had not yet developed a consistent method for writing characters. There has been no research into how these latest recognizers are doing with children as their targeted user.

The use of the neural networks as a new character recognition technique is also getting good results. With the initial success of backpropagation learning [4,25], more research has started in this neural network approach. Although not much has been done in the use of this technique toward recognition of gestures, some limited results can be seen in this thesis.

There are other approaches to character recognition, like adaptive recognizers. Adaptive recognizers refer to a class of recognizers that are able to update their prototypes in order to recognize new inputs. Such systems suffer from a stability-plasticity dilemma formulated by [46], which pointed out that when Neural Networks learn new things, they forget what they had learned before. Two current studies that try to solve this problem are Supervised Competitive Learning systems. One uses Backpropagation Networks for its learning modules [37], discussed in detail later, and the other Fuzzy Sets as its learning modules [47]. The early results of this new system are encouraging.

2.2.2. Gesture Recognition

The above mentioned systems do well in character recognition, cursive script recognition and character-like gestures [22]. However, other recognition methods are needed for non-character-like gestures, such as *delete* gesture that can differ in size, rotation and stroke order. One such method was discussed in [23], which used directions and change of directions to recognize previously unrecognizable gestures.

The latest recognizer by IBM uses a new approach to gesture recognition. Lipscomb's recognizer [31], uses a combination of angle filtering with multi-scale recognition. The multiple scales are the different angles set for the filtering. The input gestures first goes through an input filter to smooth and reduce the number of points. The results of the input filter go through the angle filter, then are tested against the prototypes for a match. If there is no match, the angle filtering and matching is tried with a larger angle. This is repeated up to five times to find a match. If no match is found, a new prototype is created. The results of the recognizer are promising for angular gestures like

the summation sign (Σ) but takes considerable training for curvy gestures like the g-clef



2.3. Related Applications

Researchers at IBM [26] investigated the use of pen-based computer (a tablet and stylus attached to a PC) in education. They used the computer to teach children how to write and had a cross word puzzle game. The same group at IBM later added a few more applications to test gestures and their use. The latest applications being a spreadsheet program, a simple sketching program, a mathematical formatter and a music score creator[3]. Their results showed that people found gestures easier and more natural than the mouse and keyboard. Another gesture application was a medical charting application [27] designed to help nurses with charting, which showed that there is a place for gestic interfaces as a user friendly and easy to learn system. Artkit [28] is a toolkit that provides gestures as a class of interaction technique along with dragging and snapping. All these applications used a limited number of gestures with very simple applications, and no information about the gestures themselves were provided. All of these early works, simply report that gestures are a successful interaction technique.

Another recent gesture system is GEdit [29,30,39], where the single purpose of using gestures is for creating and manipulating simple objects. This application has

three objects, the circle, the box and the triangle. The creation of each is achieved by drawing a line on the screen. The system provides the user with a subdivided circular menu when the pen is pressed down for a short interval of time. The direction of the line is used to determine which object is created. The system has three gestures for 'move', 'copy' and 'delete' to be used on individual objects or a group of objects. Figure 2 shows the deletion of a single and a group of objects. The 'move' command is shown in Figure 1, and the copy is achieved by completing the move gesture by drawing a 'c' at the end. GEdit is one of the few applications that is trying to learn more about the human computer interaction by studying different interaction techniques including the use of gestures. However, the small number of gestures used (three) will not give as good an indication as this thesis hopes to report with a fully gesture driven system.

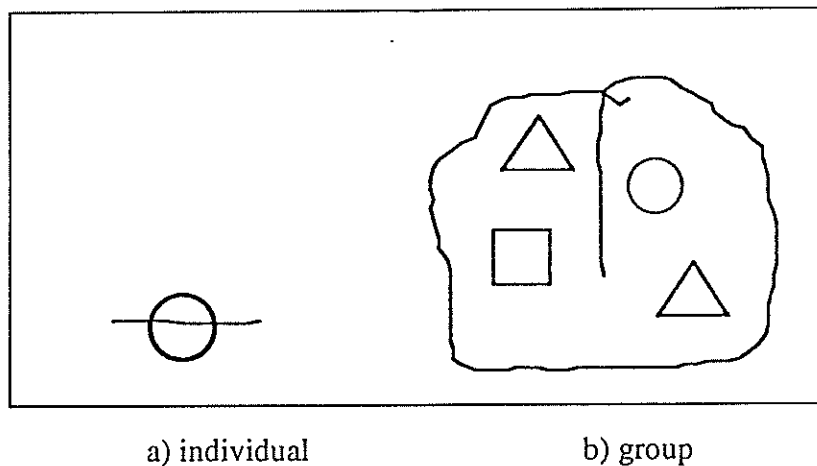


Figure 2. Deleting in GEdit

There are also trainable recognizers made specifically for gestures, e.g. Lipscomb's gesture recognizer [32] and Rubine's gesture recognizer [33], explained below. Lipscomb's recognizer is used on IBM's test pen computer platform with a spreadsheet application. The recognizer allows the training of the gestures to be done interactively using the stylus, but the hot point specification is still done using the keyboard commands. The system is stated to be adequate for system builders, but still in too early stages of development to be trained and used by end-users. IBM's recognizer looks to be a good recognizer, with good concepts at gesture recognition. However, because it is still in its development stage, the effects of this recognizer with a large set of gestures is

not stated. The reports, for now, concentrate on the recognizer's ability, rather than the issues of gesture design and gesture evaluation. This recognizer, might be another good alternate recognizer to test the gestures on in future studies.

Rubine's recognizer, named GRANDMA, is used in an example application named GDP, a gesture-based drawing program. Some of the GDP's gestures are seen in Figure 3. The emphasis is on the recognition capability. The system is able to recognize 10-13 gestures at a time. There is no explanation on how the gestures in Figure 3 were developed, or what guidelines were used. The system is implemented on X-windows with the user using the mouse to simulate the pen. The left mouse button is used for pen/down, pen/up command. This approach of using the mouse and the limited number of gestures that can be tested simultaneously limits the testing of such a complex system as we tested in this thesis, using GRANDMA as the recognizer. The draw application GDP was mainly used to show the effective recognition capabilities of the recognizer rather than testing out the usability and human interaction issues of the gestures designed.

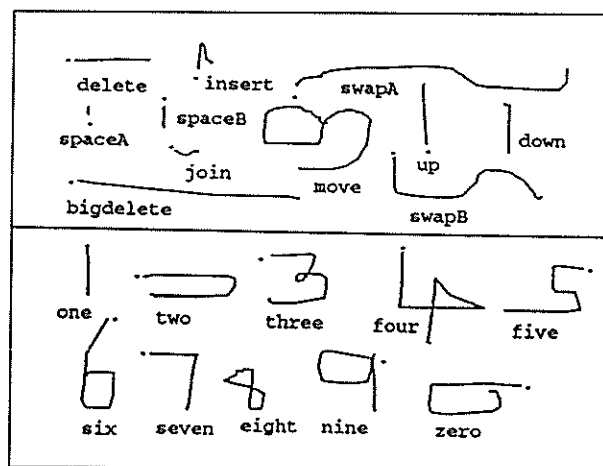


Figure 3. Different Gesture Sets used in GDP

The two commercially available pen computers employ gestures. They are the 286 based GO² prototype Computer (a.k.a. Lombard) by the GO Corporation [40], and

²GO is a trademark of GO Corporation.

PenWindows³ [41] software application by Microsoft⁴ running under Windows on a tablet. The gestures in PenWindows are few in number which helps in remembering, but the recognition capability varies from very good in 'delete' gesture to very bad for 'insert space' gesture. The small number of gestures in PenWindows also implies that the menu paradigm is still the main interaction tool for PenWindows. The gestures used by PenWindows is shown in Figure 4.

The operating system, PenPoint⁵, on the Lombard supports the use of gestures. Most gestures in PenPoint represent the menu functions available on the system. See Figure 5 for the PenPoint gestures. Some of them are based on the proofreaders marks and are used to manipulate the text. The recognizer depends on stroke order and stroke direction, along with positioning with respect to the base line and relative size to surrounding text. Some of the gestures resemble each other, like 'check' and 'new paragraph', and some are not intuitive such as the 'circle' and the 'insert space'. The large number of gestures makes them hard to remember [44].

Overall, neither one of the two commercial systems explained how they designed their gestures. Both systems are concentrating on simplicity of the gestures but not necessarily on intuitiveness of the gestures. As with the other applications mentioned before, these two systems do not report how the gestures are accepted by the users, or what principles were used to create them. In this thesis, we will not only go through the process of how each gesture was designed, but also report on how the users responded to each gesture.

³Windows is a trademark of Microsoft Corporation.

⁴Microsoft is a trademark of Microsoft Corporation.

⁵PenPoint is a trademark of GO Corporation.

PEN WINDOWS BASIC GESTURES

.	Place insertion point where you tap the pen tip. This assumes you are not entering text. (See period below.)
—	Select the information under the gesture.
✂	Cut selected text and place it on the Clipboard.
⌘	Copy selected text and place it on the Clipboard.
^	Paste the Clipboard's contents at the top point of the gesture.
↶	Undo the previous action.
—	Delete and select information under the gesture.
⌋	Extend selection to include information between the bottom point of the gesture and the previous selection.
✓	Correct word and display it in the writing window.
└	Insert space to the right of the gesture.
└	Insert new line after the gesture.
└	Insert tab to the right of the gesture.
└	Delete character to the left of the starting point of the gesture.
⊙ or circle dot.	Insert period While entering text, but before recognition, a tap (dot) is a period. A dot with a circle around it inserts a solitary period at the insertion point.













Figure 4. Pen Windows Gestures

PenPoint Quick Reference








PENPOINT

Gestures













Basic Gestures

-  Bracket left or right selects a word to its right or left. A second bracket extends the selection.
-  Bracket left or right selects a word to its right or left. A second bracket extends the selection.
-  Caret opens a writing pad or creates a document.
-  Check displays options for selected text or objects.
-  Circle opens an edit pad.
-  Cross out deletes selected text or objects.
-  Flick up, down, left, or right scrolls a line to the corresponding edge of the screen. Double flick in any direction to scroll as far as you can.
-  Insert space adds a blank space in text or boxed pads.
-  Pigtail deletes a character in text or boxed pads.
-  Press sets an insertion point or gets text or an object ready to move. Drag the marquee to the new location; it floats in case you need to turn pages.
-  Tap selects text or chooses menus and options.
-  Tap press gets text or an object ready to be copied. Drag the double marquee to the new location; it floats in case you need to turn pages.

Gestures in the Title Line

-  Borders hides the document's borders and menus. To show them again, check tap in the document and use the Access option sheet.
-  Cork margin shows or hides the cork margin.
-  Double tap floats a document. It also works on tabs and page numbers in the Table of Contents.
-  Find searches for a specific word in a document or selection, and allows you to replace it.
-  Flick up or down to zoom or reduce the page view, if that preference is set.
-  Menu shows or hides the document's menu line.
-  Tab adds or deletes a document's Notebook tab.

Editing Text in a Document

-  Bold makes words or selected text bold. Italics I italicizes them. Underline U underlines them. Normal N makes them plain.
-  Caret tap opens a large embedded writing pad.
-  Circle tap creates a hyperlink button.
-  Double caret creates an embedded document.
-  Double tap selects a word. Triple tap selects a sentence. Quadruple tap selects a paragraph.
-  Insert character opens a box for adding a single letter.
-  Line through deletes characters in edit and writing pads.
-  New paragraph inserts a paragraph break.
-  Proof displays alternate spellings of a word.
-  Scratch out deletes words. It also works in writing and edit pads, and in fields.
-  Spell checks the spelling of a document or selection.
-  Upper case formats text as all capitals. Initial caps capitalizes words. Lower case formats text without capitals.

©1991 GO Corporation. All rights reserved. PDEV#044-00007

Figure 5. PenPoint Gestures

3. Box and Arrow Editor (BAE)







BAE was designed as a test-bed for a preliminary investigation into designing gesture systems for pen computer applications. The Editor serves no practical purpose, except that it provides us the insights and experiences of developing a box-arrow based visual programming language for pen computers, such as Hyperflow.

Through this editor, one creates and manipulates *box-graphs*. A box-graph is a two dimensional, single page display on a computer screen that contains boxes and arrows. The user can save into and later retrieve these box-graphs from a file.

3.1. Objects

The box is a rectangle of any size. There are three types of boxes with outlines of different widths and patterns. See Table 1 for the types. The arrow is defined as a straight line of any length. The arrow, like the box, has the same three types of outline. The arrow can also have one, two, or no arrow heads to define directionality. The arrow head is defined as a fixed sized, solid triangle, pointing in the same direction as the axis of the arrow. Since this editor is specifically designed as a test bed for gesture design and has no practical use, it has no semantic content, except for the 'duplicate' command, which examines the enclosed area of the targeted box for any arrow end points. Any size and any placement of boxes and arrows are allowed. Figure 6 shows a sample box-graph. The choice of objects to be included in BAE is motivated by the design of Hyperflow.

Table 1. Description of Object Types

Type	Box	Arrow
Normal		
Thick		
Dashed		

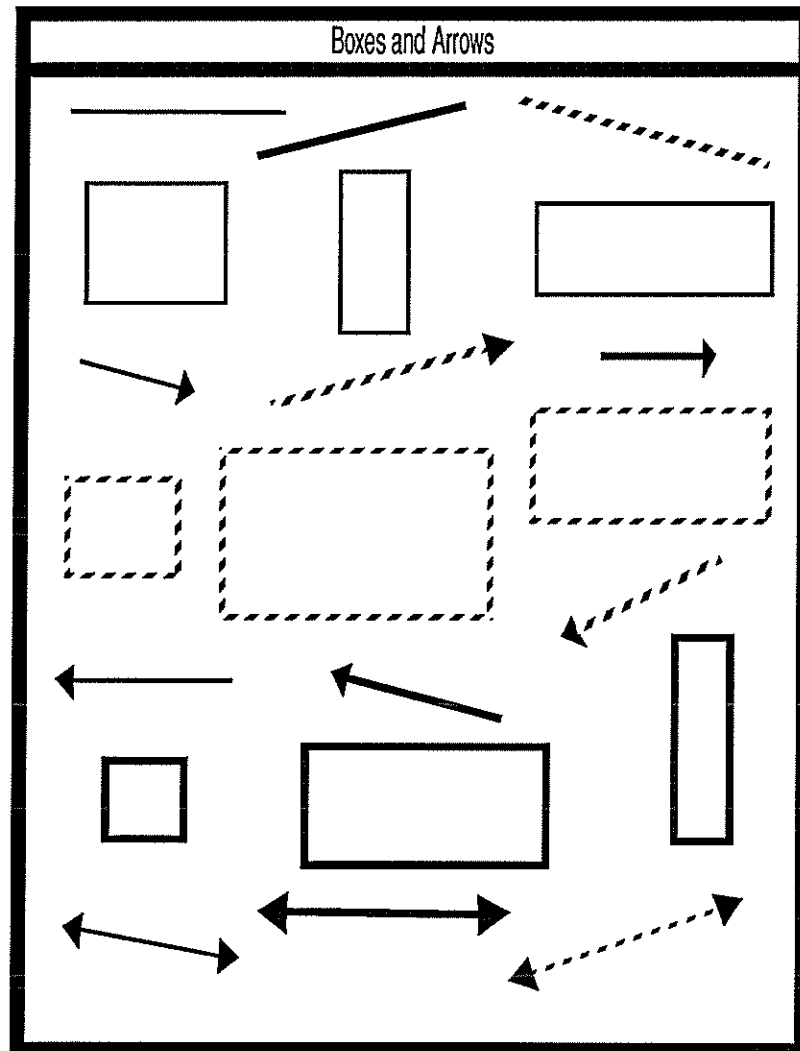


Figure 6. Sample box-graph

3.3. BAE Commands

BAE users manipulate box-graphs through a sequence of commands. Each command is represented by a gesture. BAE's 43 commands can be separated into five groups; documentation, editing, formatting, tools, and miscellaneous. A detailed explanation of each command is given in Chapter 4 when their gestures are described. A brief description of each group follows:

Documentation commands enable the user to open and save the box-graphs into files. In addition, he/she can create new box-graph files and close the existing file.

Editing commands perform the standard edit functions, such as cut, duplicate, copy, paste, and move, on the selected objects. Another command from the editing group, is the ability to select objects. The user can select or deselect objects one at a time or all of them at once.

Formatting commands deal with the manipulation of selected objects. From resizing objects to alignment on the screen. The formatting command allows the user to turn on the grid for more precise resizing. The rest of the formatting commands deal with the displaying of the Grid Panel, showing and hiding of the grid on the screen. Using the Grid Panel the user can change the width and height of the grids spacing. The alignment commands are: align top, align bottom, align left, align right align vertical, and align horizontal.

Tools commands are used for the creating the boxes and the arrows.

Miscellaneous commands are 'quit', to exit the program, and 'help', which gives the user a list of all commands available, along with their respective gestures.

4. Gesture System for BAE

There are some common sense guidelines for designing gestures. Here we will state them as Requirement Specifications for Gesture Design. The requirements are simplicity, drawability, complexity, rememberability, and intuitiveness. Simplicity, usually, refers to the number of strokes in the gesture, with one being ideal. Some of these requirements are related to each other. If the design is simple, it usually is easy to draw. Most easy to draw gestures are also easy to remember. By complexity we mean both the gesture itself (the physical attributes) and its hot point placement. Hot points that are placed at the beginning or ending or at the very middle help with complexity and rememberability. The intuitiveness of the gesture is the most important gesture quality. Intuitiveness helps rememberability and complexity of the gestures. If a gesture is not intuitive, no matter how simple and easy to draw it is, it will not be remembered easily.

In our system, because we concentrated in designing the gestures, we did not develop a recognizer, but used an available character recognizer with some code added to deal with special characteristics of gestures. This approach to implementation however brought a very important problem to the surface. Because we used a recognizer which was originally designed for characters, the recognition was performed using features important in characters, but not necessarily important in gestures. In fact the results of the recognition testing clearly showed this problem, discussed further in chapter 6.

In view of such an approach, the designer needs to consider the recognition problem, i.e. the designer needs to know beforehand how the recognizer works. Knowing the strengths and weaknesses of the recognizer will contribute significantly to the design of each gesture. Other important features that the designer needs to address about the recognition process is how and where to deal with hot point and closure problems.

The BAE gesture syntax must satisfy these requirements for each gesture we design. A detailed explanation for the gestures of each command is given later in this chapter.

The first requirement we needed to face was simplicity. Easiest way to achieve this is to make sure all the gestures used are single stroke gestures. However, not all single stroke gestures are necessarily intuitive. Because of this, we placed a greater importance in making gestures intuitive rather than enforcing the single stroke requirement.

Secondly, some of the commands need more information to be fully executed, such as the file management commands. In this study we concentrated on the top most layer of interaction with the user. The second layers that gather more information from the user are still entered with the commonly seen menuing methods. In a fully gesture driven system such second layers would also be converted to operate with gestures. Examples of these secondary layers are the Open, SaveAs and Grid Panel Commands which produce input windows to gather information.















There are thirty one gestures in BAE listed in Figure 7, ranging from lower case letters used for documentation functions to using 'T' in different rotations to indicate alignment of the object.












The reasoning for having so many gestures, is to test if it is possible to create a drawing application software that would totally avoid the use of menus, by replacing all commands with gestures. The selection process started by examining drawing applications, like Draw [42], and TopDraw[43] to generate a list of common commands for such applications. From this list the 43 commands were selected.

During the design of these gestures, we separated the gestures into three groups, based on origin. We termed the first group as Alphabet Oriented Gestures. These gestures are the first letter of the command name we use to execute, in our case, the

document commands. This is to test if there is a mental connection between certain letters of words and with the specific operation. Like 'o' for open when performing an open box-graph command, or 's' for save when saving a box-graph.

Commands within working window

-  Cut/Grid Off - If on an object deletes it. If grid is visible, turns grid off.
-  Copy/Close - Places a copy of the object in to the buffer, or close the working window.
-  Duplicate - Produces another object having the same characteristics as the source object.
-  Paste - Places a copy of the object in the buffer to the specified location.
-  Grid Panel - Displays the grid scaling window.
-  Align Right - Aligns the object selected on the right border
-  Align Left - Aligns the object selected on the left border.
-  Align Top - Aligns the object selected on the top border.
-  Align Bottom - Aligns the object selected on the bottom border.
-  Align Vertical - Aligns the object selected in the middle between top and bottom borders.
-  Align Horizontal - Aligns the object selected in the middle between right and left borders.
-  Normal Box - Creates a box with regular line width.
-  Thick Box - Creates a box with thick line width
-  Dotted Box - Creates a box with dashes as its outline.

-  Normal Line - Creates a line with regular width.
-  Normal Left/Right Arrow - Creates a unidirectional arrow.
-  Normal 2-headed Arrow - Creates a bidirectional arrow.
-  Dotted Line - Creates a line with dashed lines.
-  Dotted Left/Right Arrow - Creates a unidirectional arrow with dashed lines.
-  Dotted 2-headed Arrow - Creates a bidirectional arrow with dashed lines
-  Thick Line - Creates a line with a thick width.
-  Thick Left/Right Arrow - Creates a unidirectional arrow with a thick line.
-  Thick 2-headed Arrow - Creates a bidirectional arrow with a thick line.
-  Quit - Quits the program, with a warning if not saved.
-  Help - Displays a help window containing a list of gestures and simple explanations.

Commands on Title Area






-  Open - Brings Up a Menu for file selections
-  Save/Save As/Show Grid - Either saves or brings up menu for new file name
-  New - Opens up a blank working window


Figure 7. BAE Commands and Gestures

The second group of gestures are the standard proof reading marks, like  for delete and  for insert (for draw its paste). The feasibility of such familiar commands to be used as gestures have been thoroughly investigated [11]. These investigations found good results with the use of these markings in editing text. We wanted to see if such editing marks could also be used to edit graphical objects.

The third group of gestures, are Artificial Gestures. These are gestures that have no previous syntax or semantics associated with them. These are used for some of the formatting commands and for the tools commands.

4.1. Formatting Command Group


The alignment commands for right, left, top and bottom are artificial gestures that look like a capital 'T' at different orientations. The idea for this came from using a line to represent the different sides of the document. We wanted to have a horizontal line to represent both the right and left alignment. A simple way to distinguish which was which, was to place a line in the middle of the horizontal line on the side that the objects

would be aligned on.  gesture then represent a left alignment. The same idea was

used to obtain similar gestures for the remaining three alignment commands. The two special alignments, horizontal and vertical, also got their shapes from the same idea. Each was obtained by combining the two alignments related to it. For example, horizontal align is aligning between right and left edges, thus combine both left align and right align to get two vertical lines and a horizontal in between. Thus the vertical gestures is obtained by combining the top and bottom align gestures.

4.2. Tools Command Group

The tools command group is the tools gestures used to create the boxes and the

arrows. First the use of the  for the box. This had two reasons for it, the ability to

define the size of the box with the user entering the height and width, and its simplicity. It is a one stroke gesture and only takes half the distance and time than drawing a box with all four sides. The arrow gestures were developed by trial and error on a piece of paper. Trying to see what possible ways are there to reducing this two stroke gesture ">" into one stroke and still be intuitive. The secondary part of both arrows and boxes gestures are what defines their type. If the user just draws the gesture then the object is solid. If the gesture is followed by a tap, then the object is dashed and finally if the gesture has a retrace on the ending part of it then the gesture is of type thick. Again this was obtained by trial and error, examining how to make thick lines with a pencil (a lot of retracing over the line) thus the idea of doing a partial retrace on the ending of the gesture. The dashed objects was a little harder, since the more intuitive way was to do the gesture in dashes, but this defeated the single stroke, simple stroke concept. The end result was to add a second stroke but make it very simple one, like a lone tap.

The same trial and error method was used to derive the rest of the gesture for the BAE. Following is the complete gesture list, along with a description of each gesture syntax and its semantics.

The following format will be followed for each command.

Gesture Command Name Picture of Gesture

Gesture Definition -

Syntax -A presentation of syntactic properties is given, such as rotation, size, stroke order, stroke direction, stroke number, physical attributes necessary for successful recognition, and hot point of the gesture.

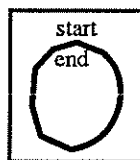
Semantics -Then meaning of the gesture is described

Note: If the input starts *within* or *on* a selected object (box or arrow), the Editor will not recognize this input as a gesture but will interpret it as the move or resize command.

Some gestures must be entered in a specific part of the screen. The screen will be separated into three different regions, the working window, the title region, and the document commands region. All commands that work on the title region also work in the document region. The screen layout of BAE is shown in Figure 18 in Chapter 5, along with a more detailed explanation.

Document Commands

Open, New, Save, SaveAs and *Close* make up the document commands. These commands are used to manipulate the box-graph files from with the BAE application.



Open

Syntax - Scribe an "O" anywhere in the title region of screen to execute this command. It is a single stroke gesture, drawn in a counter-clockwise circle without a hot point. The end points should be connected for better recognition results. The document command region is also available for this gesture.

Semantics - If there are no box-graph files with .bae specification in the current NeXT directory, the command creates a working file with UNTITLED.bae as the default title.

If at least one file exists, then the user is presented with a window containing the list of graphical file names (presented 5 at a time) appearing on the lower part of the screen. If more than that are in the directory then page "up" and page "down" buttons become available. The window also contains the buttons for "cancel", "open" and "new". A single tap with the stylus within the box containing a name selects that file. If open command is performed, with an already existing box-graph, then the user is given a chance to save the current box-graph before the open command window (see Figure 8) is displayed. Any one of the three commands 'cancel', 'open', and 'new', appearing on the right of the window will cause the window to be erased and the command executed. A gesture driven alternative would have been to only display the file names and let the user use gestures for cancel (maybe a cut gesture), open (double tap on file) and up and down lines for scrolling the other file names.

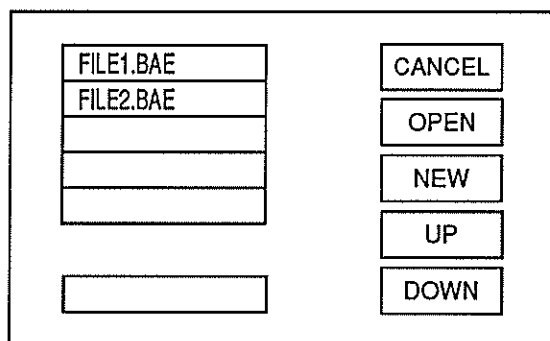
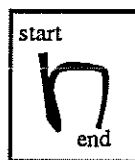


Figure 8. Open Command Window

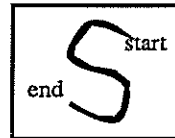


New

Syntax - Scribe an "n" anywhere in the title region of the screen to execute this command. It is a single stroke gesture without a hot point, that should be drawn fairly straight up, i.e. it should be rotation sensitive. Stroke should start upper left go down to lower left then going up, then to the right and then down. The document command region is also available for this gesture.

Semantics - Creates a working box-graph with UNTITLED.bae as the default title if no

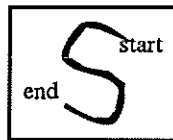
changes since last save, else a window similar to Figure 11 is displayed on the lower part of the screen with "cancel", "save and new", "don't save but new" options. The user can then save the present box-graph before opening the new box-graph or just open a new box-graph.



Save

Syntax - To save a file, one uses a single downward gesture, to scribe an upright "S" either in the title region of box-graph - except where the box-graph title is, or in the document command region.

Semantics - Saves the working box-graph with the given name to the current directory. Box-graph files are saved with ".bae" ending. If the box-graph was new and Save As was not invoked, then its saved with the default name of "UNTITLED.BAE".



SaveAs

Syntax - To use the saveas command, follow the same directions as in the save gesture, with the starting point within a very close vicinity of the box-graph title in the title region.

Semantics - The user is presented with a window containing the default name (Figure 9). The window also contains letters A-Z and 0-9 and a few more punctuation marks for the user to make up their own graphical file name by tapping it out. The graphical files are saved with .bae ending. There is a 16 character limit to the box-graph title. Again an alternate, all gesture solution would have been to use a character recognizer, and have the user write the name of the file and scribe an 's' to save.

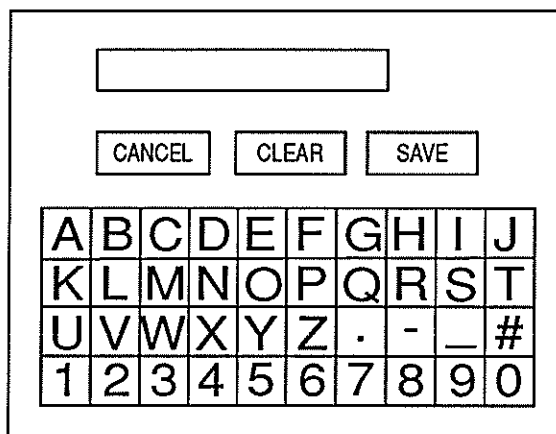
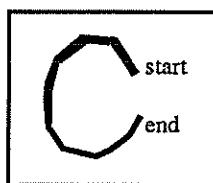


Figure 9. SaveAs Command Window



Close

Syntax - Close command can be achieved with a single downward gesture to scribe a "C" anywhere in the title region of screen or document command region. It has no hot points, with the end points not near each other.

Semantics - A window is displayed with either the commands "cancel" and "close" (Figure 10) or if no changes since last save, options to close with or without saving the box-graph (Figure 11).

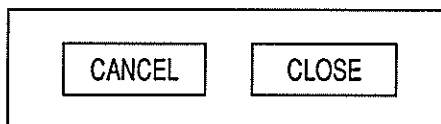


Figure 10. Close Command Window (with Save)

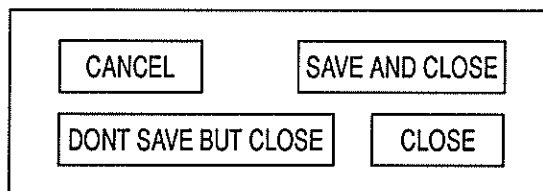
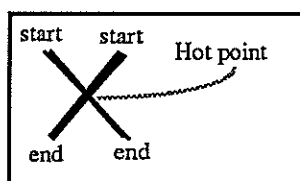


Figure 11. Close Command Window (without Save)

Edit Commands

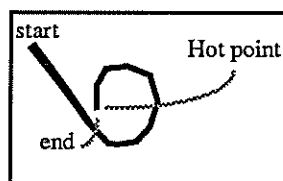
The edit commands are, *cut*, *duplicate*, *copy*, *paste*, *select and move*.



Cut

Syntax - To delete an object, simply cross (X) out the object. The object containing the intersection point of the two lines (the hot point) forming the x will be deleted. Stroke order and stroke direction do not matter in its recognition.

Semantics - The object is deleted from the box-graph and a copy of the object is placed in the cut-paste buffer. The copy in the buffer is only the object selected, none of the connections (if any) are saved. Connections (arrows) are simply lines that have an end point within the selected box. If cut command is used with multiple objects selected, then the items are deleted but no object is placed in the buffer.



Duplicate

Syntax - Draw a line to the location of the new object, starting within the object to be duplicated and ending the line with a semi-circle. The new object will be centered around the hot point, the end point of the gesture. A single stroke gesture. The line should start upper left and go lower right, then start the half circle in the end going counter clock wise.

Semantics - A new object is created with exactly the same specifications as the original. If the object is a box, the new object will also be connected in exactly the same manner to all other boxes as the original was. See Figure 12 for this special case duplicate. Note that using duplicate, the new duplicated boxes will be connected in the same manner as the original.

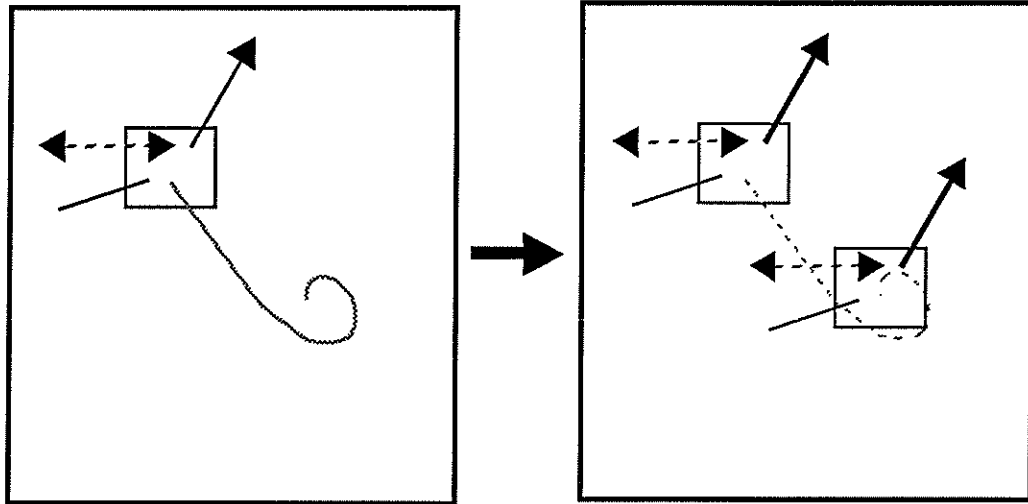
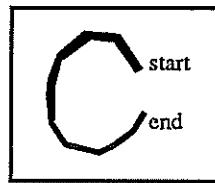


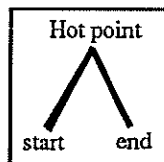
Figure 12. The Duplicate Command



Copy

Syntax - To copy an object into the buffer, use a single downward gesture to scribe the character 'C' within the object. The hot point is the starting point of "C", which *must* start within the object chosen to be copied.

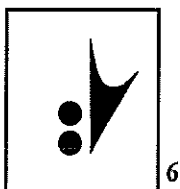
Semantics - A copy of the object is placed in the buffer. Just like cut function, this function only saves a copy of the object - this does not include any of the connections it might have with other objects. This command does not work with selected objects.



Paste

Syntax - Available after either a copy or a cut command has been used, which determines the object in the buffer. The gesture is the carat (^) or upside down v - with the stroke starting at lower left going to the right and up, then to the right and down. The tip of the carat is the hot point determining the location of the new object (centered around the hot point).

Semantics - A new object is created which is exactly the same as the object in the buffer. This does not include connections. Note that the only way to create objects with their connections is to perform a duplicate.



Select

Syntax - There are two ways to enter the select command.

First method. A double tap, a successive touching of item by pen within a short interval, within or on the object of interest. Currently the second tap must follow within a half second.

Second method. Hold down the pen for a pre defined amount of time within or on the object to be selected, currently set at 1.5 seconds.

Semantics - The purpose is to bring to the front an object to be manipulated on. After the selection of the object, the knobs will appear, highlighting that object, verifying its selection.



Incremental Select

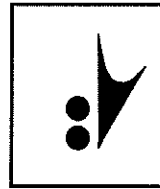
Syntax - Select the first item with a double tap. A single tap within or on an unselected object will select that object.



Incremental Deselect

Syntax - The tap within or on the object acts like a toggle, thus to deselect a selected object just tap within or on it.

⁶This notation for taps is the meta language used by GO Corporation.



Select All

Syntax - A double tap, within the title region of the screen.

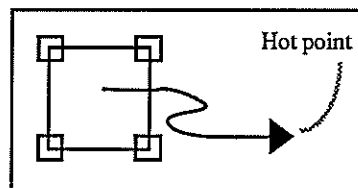
Semantics - When this command is performed, all the objects within the working box-graph become highlighted - i.e. the manipulation knobs appear on all the objects.



Deselect All

Syntax - A single tap anywhere in the box-graph not containing a object will perform a deselect all.

Semantics - This command simply removes the manipulation knobs from all the selected objects, i.e. it deselects all the objects.



Move

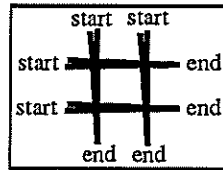
Syntax - There are two ways to move an object.

First method. *Select the object first.* Starting within or on the object selected drag to the new place. The hot point of this gesture is the end point of the dragging. A single stroke gesture after the determination of the object.

Second method. Starting with in, or on the object, hold the pen for a predefined amount of time (thus selecting the object), then drag to the new location. The hot point is again the end point of the dragging.

Semantics - The object is in the new position now. Move only moves a single object.

Format Commands

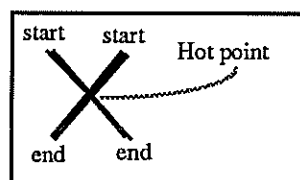


Grid Panel

Syntax - It is the pound sign drawn within the working window region. It should be free of rotation or size, and the stroke order or direction should not matter. The four intersections are necessary for better recognition. The two set of parallel lines need not be perfectly perpendicular but should check for close parallelism.

Semantics - The execution of this command brings to the foreground a window that shows the current spacing settings for the grid system (expressed in pixels). The user can then change and set the spacing to new dimension. The spacing is limited to be between 10 and 100 pixels. See Figure 13. Again, an alternate solution would have been to use a character recognizer and have the user write the spacing.

Figure 13. Grid Command Window

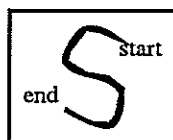


Turn Grid off

Syntax - When the grid is visible on the box-graph then a cut gesture "X" anywhere

within the working window region (not over an object) will turn the grid off. A word of caution: with the grid on it is not possible to select multiple objects and perform a cut operation to delete them. With the grid on, the cut gesture not on an object will be interpreted as turn grid off.

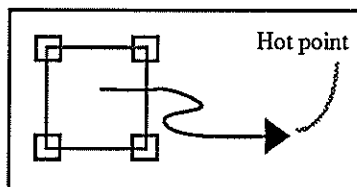
Semantics - This command is in effect when the grid panel is visible. This command effectively makes the grid invisible.



Show Grid

Syntax - Scribing an "S" on the working window region of the screen displays the grid on the box-graph. Placement of the "S" is unimportant.

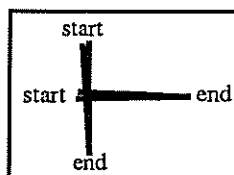
Semantics - The grid is made visible.



Align to Grid

Syntax - This is the same as the 'move' gesture. The object to be aligned has to be selected. Once the object is selected, the move gesture is used to snap the object on to the grid.

Semantics - Once the grid is visible the manipulation of the knobs are interpreted as snap onto grid commands. The direction of the drag of a knob determines which edge or edges are snapped onto the grid.



Left edge Align

Syntax - First select the object(s) to be aligned. Then enter the gesture. The gesture looks like a capital "T" on its side. However, its stroke order and stroke direction are very specific to clearly distinguish it from horizontal or vertical align commands, which

look similar. The vertical stroke is performed first, going from top to bottom, then the horizontal stroke, going from left to right, with the two lines intersecting for better recognition. The angle between the two strokes should be within a predefined margin of ninety degrees, i.e. as close to perpendicular as possible. The horizontal stroke should not start too far left of the vertical stroke as to make the stroke look like a "+" or a "t". The intersection of the stroke should be in the middle section of the vertical line.

Semantics - The selected objects are redrawn with the left most points flush with the left most point of the left most object selected. Flush against the left wall of working box-graph if only a single object is selected. See Figure 14.

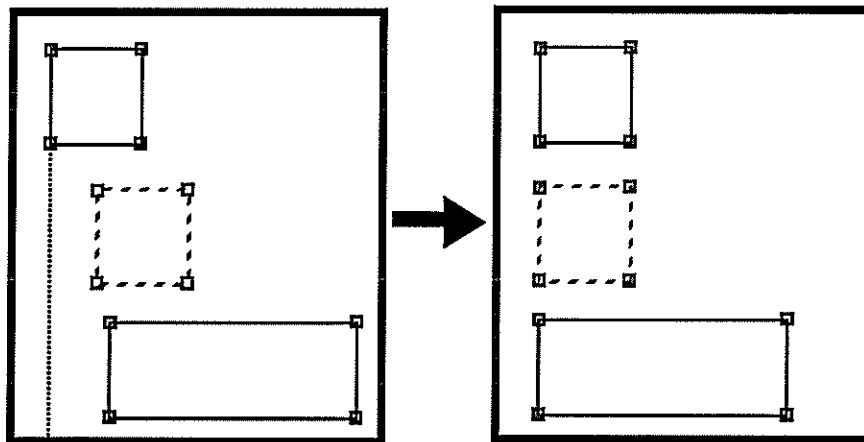
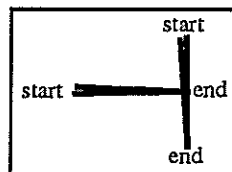
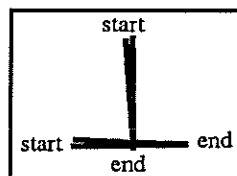


Figure 14. Align Left Function



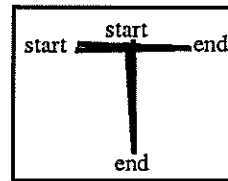
Right edge Align

Syntax Semantics - Exactly the same as left edge command, but alignment to right most point, or to right wall.



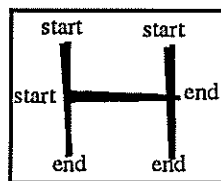
Bottom edge Align

Syntax Semantics - Exactly the same as left edge command, but alignment to bottom most point, or to bottom wall.



Top edge Align

Syntax Semantics - Exactly the same as left edge command, but alignment to top most point, or to top wall.



Horizontal center Align

Syntax - Select the object to be centered, then using three strokes, scribe a capital "H". The lines should intersect for better recognition.

Semantics - The selected object(s) is redrawn with the left most point being the same distance away from the left edge of the working box-graph as the right most point is from the right edge of the working box-graph. See Figure 15.

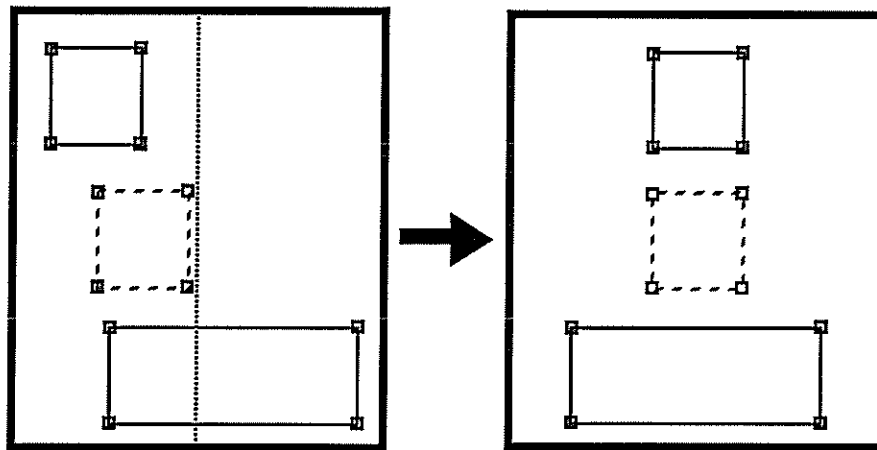
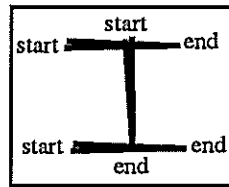


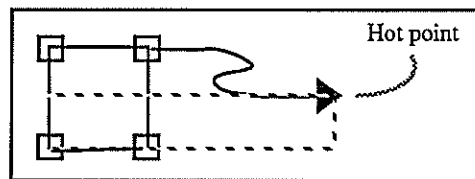
Figure 15. Align Horizontal Command



Vertical center Align

Syntax - Select the object to be centered, then using three strokes, scribe a capital "I". The lines should intersect for better recognition.

Semantics - The selected object(s) is redrawn with the top most point being the same distance away from the top edge of the working box-graph as the bottom edge point is from the bottom edge of the working box-graph.



Smaller Box

Syntax - Select the box, then use the knobs for resizing.

Semantics - The object is first selected, then the knobs are manipulated in the desired direction(s) to achieve the desired size.

Larger Box

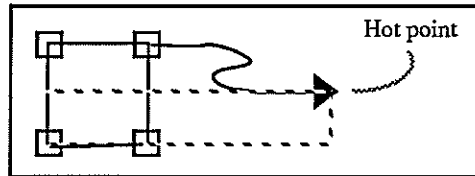
Syntax Semantics - Similar to Smaller Box command

Longer Arrow

Syntax Semantics - Similar to Smaller Box command

Shorter Arrow

Syntax Semantics - Similar to Smaller Box command.



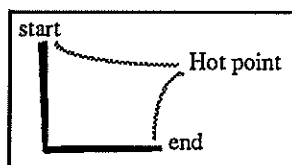
Size to Grid

Syntax - The object to resize is selected. Then the knobs are moved to correctly snap the object on to the grid.

Semantics - The object selected can be made to size to grid by snapping the knobs onto the grid, by simple drags of the corner knobs.

Tool commands

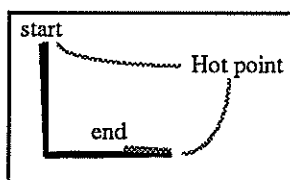
There are two different kinds of objects in this editor, boxes and arrows. The creation of both are described in detail below. Manipulation of these objects are explained in the Format Commands section above. Figure 6 shows all the possible objects within BAE.



Normal Boxes

Syntax - This is a single stroke gesture, with the stroke going from top to bottom, then to the right forming a "L" shape. Notice the only difference between this gesture and the thick box is the third section of the stroke with going over the horizontal section. The hot points are the first point of contact, and the last point of contact, defining the opposite corners of the box.

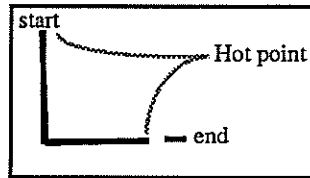
Semantics - This will create a box with normal lines



Thick Boxes

Syntax - The 'L' shape of the normal box gesture is drawn, then without lifting the pen, go back over (at least a third) of the horizontal section, just drawn. Notice the only difference between this gesture and the normal box is the third section of the stroke with going over the horizontal section. The hot points are the first point of contact, and the farthest right point established with the horizontal strokes, defining the opposite corners of the box.

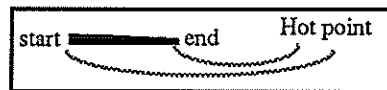
Semantics - This will create a box with thick lines.



Dashed Boxes

Syntax - This gesture is identical as the normal box gesture, with the addition of a single tap within a specific amount of time of ending the "L" gesture. The tap should be within the near vicinity of the end point of the "L" stroke. The hot points are the first point of contact, and the last point of contact of the first stroke, defining the opposite corners of the box.

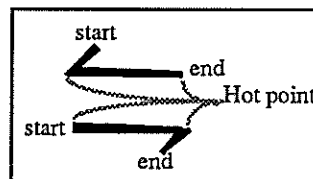
Semantics - This will create a box with dashed lines.



No-headed Arrows

Syntax - This gesture is simply a straight line. To add the type of arrow, i.e. thick, normal or dashed see appropriate directions below.

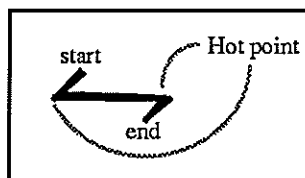
Semantics - This will create a line between the first and last point of contact.



One-headed Arrows

Syntax - These are a single gestures. If the arrow head is to be drawn first (the left arrow), then the line starts to the right and above the intended tip of the arrow, then drawn to the tip and then complete the arrow body. The angle created in the stroke should be between 30 to 60 degrees, to clearly look like an arrow head. Again, to add the type of arrow see directions below.

Semantics - This will create either a left or a right arrow, depending on when the arrow head was drawn.



Two-headed Arrows

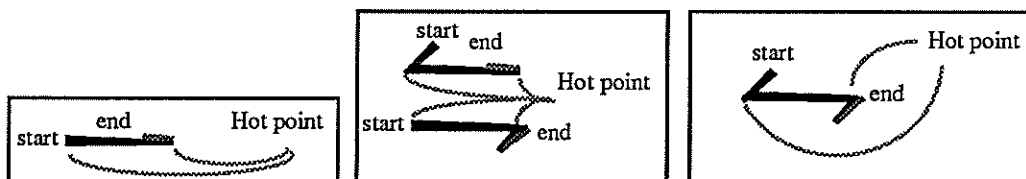
Syntax - This gesture start the same way as the single headed arrow (the left arrow), but rather than stopping upon ending the arrow body, the line is continued away from the arrow body forming an angle between 30 to 60 degrees(like the ending of the right arrow). The ending stroke should not be too short to avoid looking like a "hook". Again, to add the type of arrow see directions below.

Normal Arrows

Syntax - If nothing is done after the main arrow gesture, then a normal arrow will be created.

Semantics - The arrow produced is of normal type.

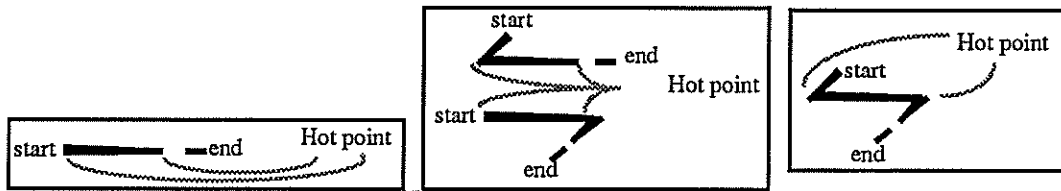
Thick Arrows



Syntax - So far the gesture only specifies the directionality, to add the thickness complete the arrow gesture by going back over the last drawn segment.

Semantics - A thick arrow is created.

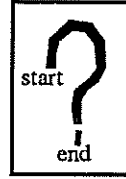
Dashed Arrows



Syntax - The dashed arrow is achieved by first drawing the main arrow gesture for the desired arrow (as described above), followed by a single tap, within the vicinity of the end point of the arrow gesture.

Semantics - A dashed arrow is created.

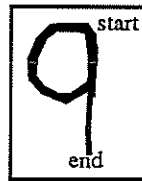
Miscellaneous Commands



Help

Syntax - Scribe a "?" on the title region of screen. A single stroke gesture is sufficient, however adding the period assures a higher recognition. The stroke direction is from upper left, circling up and to the right then down. There are no hot points in this gesture.

Semantics - This command will pop up a window that contains the list of all available commands with a small description of each, and the gesture that performs that command.



Quit

Syntax - Scribing a "q" anywhere on the screen will quit the program. It is an upright gesture without any hot points.

Semantics - This command effectively quits the program. This command will pop up a window just to make sure the user did intend to quit and the command was not done in error. See Figures 10 and 11.

5. Implementation

In order to evaluate the quality (effectiveness) of the gesture designed for BAE, a version of BAE was implemented on the NeXT⁷-Lombard⁸ system.

5.1. BAE System Setup

The editor is implemented on the NeXT Workstation connected to the Lombard prototype machine. The NeXT Workstation has a 68040 CPU, a 105 MB hard disk and 8 MB RAM, and runs NeXT Software Version 2.0. Lombard is a 286 based pen computer with an 8 MB RAM running DR version of PenPoint SDK⁹ Operating System.

The two machines are connected via a serial line as shown in Figure 16. The Lombard is responsible for capturing all the pen data (pen status of 'down/up' and coordinates of pen location when the pen is down), and for inking when the pen is down, and erasing the ink when the pen is lifted. The Lombard also displays all objects. The NeXT receives the pen data (gesture information), and interprets the gesture commands. It keeps track of the data structures representing a box-graph. The NeXT sends draw commands (Table 2) to the Lombard to draw and erase all objects and to display all other information on the Lombard.

⁷NeXT is a trademark of NeXT Computer, Inc.

⁸Lombard is a trademark of GO Corporation

⁹PenPoint SDK is a trademark of GO Corporation

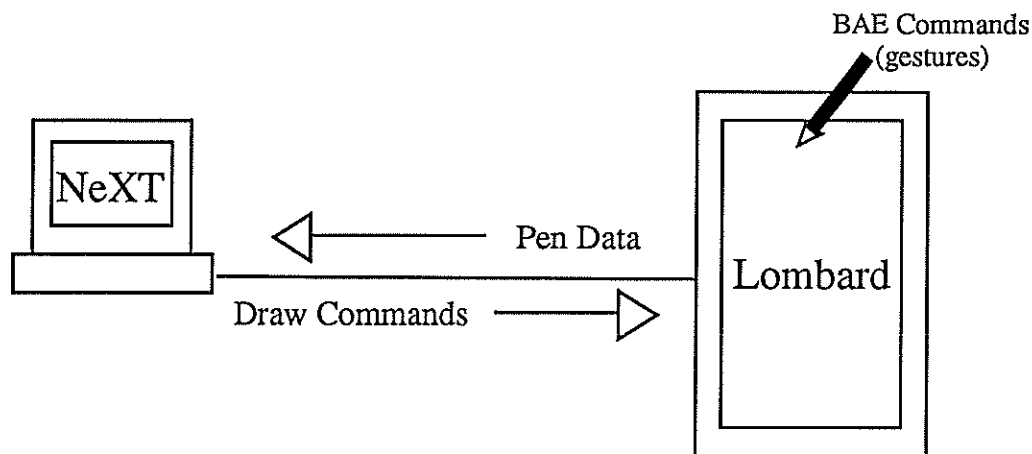


Figure 16. Lombard NeXT Hardware Connection

The software on the Lombard was written in Pen Point Operating system¹⁰. The software on the NeXT was written in C on the NeXT platform. GNU C compiler was used to compile and link all the files for BAE and the recognizer written in Objective-C on the NeXT platform. The Lombard code is approximately 1000 lines of C code producing a 36K sized executable. The Editor code (without the recognizer) is 5500 lines of C code producing a 123K sized executable. The gesture recognizer is 450 lines of C code producing a 5407 byte .o file which is linked with the Editor's .o files to produce the 123K sized executable. Serial port runs at 9600 Baud. The Lombard screen size, used for scanning BAE gestures, is limited to 590x380 of (640x400). The digitizer on the Lombard scans 100 points/second, which it then transmits over the serial port to the NeXT.

Figure 17 shows the software architecture for BAE. Descriptions of each unit is stated below.

5.2. NeXT Software

The software units besides the Gesture Recognizer were needed since not all the operations were trained to be recognized by the Recognizer. The Move, Resize,

¹⁰The software on the Lombard was written by Dale Frye.

Selection Deselection operations are handled separately. The First Point Detector checks in its data structures to see if the initial point is on or within a selected object. The Tap detector checks all the points of the input to see if it satisfies a tap. The Selector/Deselector, depending on the tap being outside or inside of object, selects or deselects one or all the objects. The Move/Resize perform the move and the resize operations on the selected object, whenever a none tap within a selected object is detected. The Gesture Recognizer is the trained Neural Network that recognizes the input and gives the result to the Editor to process. The algorithm that the Gesture Recognizer uses is described in section 5.4. The Editor processes all the recognized gestures along with printing messages to the Display and performing hot point detection and closure.

The BAE program runs on the NeXT computer, reading the pen data collected by the Lombard from the serial line. The NeXT software processes all the points and sends commands back to the Lombard on the serial line, instructing it with which primitives need to be executed. The NeXT software uses simple linked list structure to keep track of all the objects.

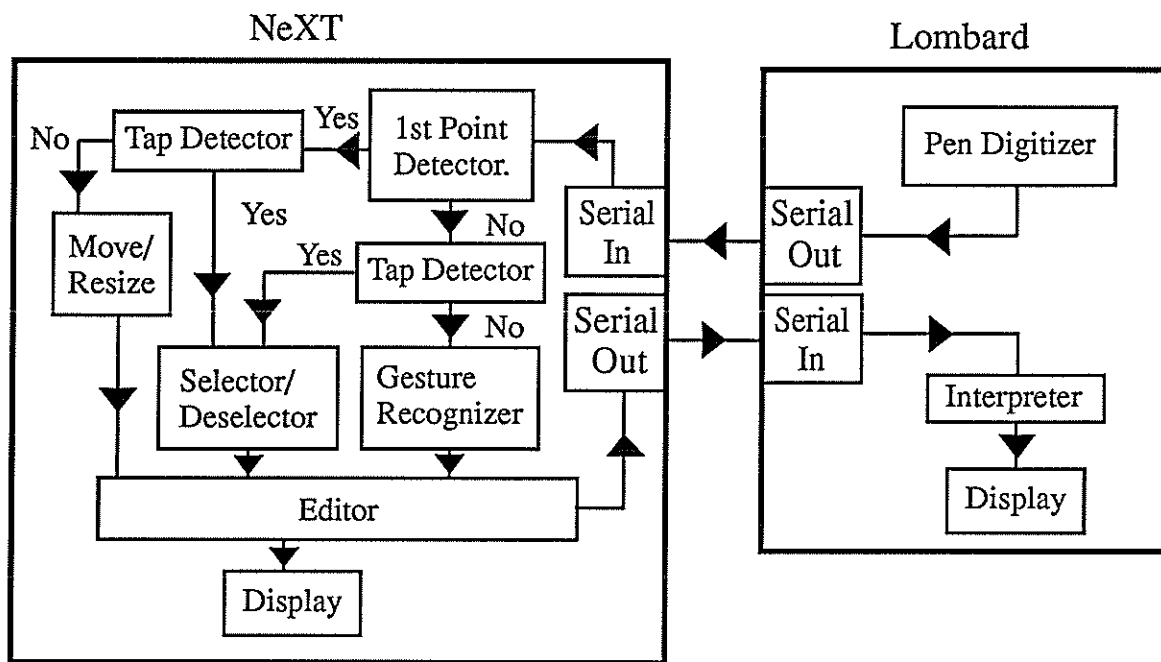


Figure 17. Software Architecture

5.3. Lombard Software

The software on the Lombard system is responsible for all the drawing needs on the Lombard machine. The Interpreter converts the draw commands from the NeXT into actions on the Lombard screen. It acts as the manager for these commands, such as drawing and erasing boxes and lines of different sizes, shapes and boundaries, erasing of regions and printing text. A two-tone bell was also made available as a command.

The Pen Digitizer handled scanning for all inputted pen values and inking. See Table 2 for a complete listing of draw commands developed on the Lombard system for BAE.

Table 2. Draw Commands

Command Name	Command	Line Type	Coordinates	String
Rectangle	R	Yes	2	No
Line	L	Yes	2	No
One-Headed Arrow	O	Yes	4	No
Two-Headed Arrow	T	Yes	6	No
Erase Area	E	No	2	No
Grid	G	No	2	No
Clear Screen	C	No	No	No
Beep High	BH	No	No	No
Beep Low	BL	No	No	No
String	S	No	2	Yes

A typical command from NeXT to Lombard is: "R 1 100 100 200 200" - to display a rectangle with lower left coordinates of 100,100 and upper right coordinate of 200,200. Table 2 shows that commands like Beep High consists of only the command byte, since no other information is needed. For example, the two-headed arrow required 6 sets of

coordinates: 2 coordinates indicating the starting and ending points of the arrow, while the other sets of 2 coordinate pairs define the triangle for the arrow head, giving a total of 15 bytes including null bytes used for spacing. The string command passed the starting coordinate of the string, then the string itself.

5.4. Screen Layout

The Screen Layout of BAE on Lombard is shown in Figure 18. It consists of three regions: Title region, working window and document gesture region. The working window shows the complete box-graph. The creation and manipulation of objects takes place in this region. The different regions also account for region sensitive gestures. The document gestures are not executed within the working window. The SaveAs gesture must be made over the box-graph name in the title region. The Save/SaveAs/Show Grid gestures are one and the same gesture, the region in which it is drawn determines which command is executed. The same is true with Close/Copy command gesture and Cut/Grid Off command gesture.

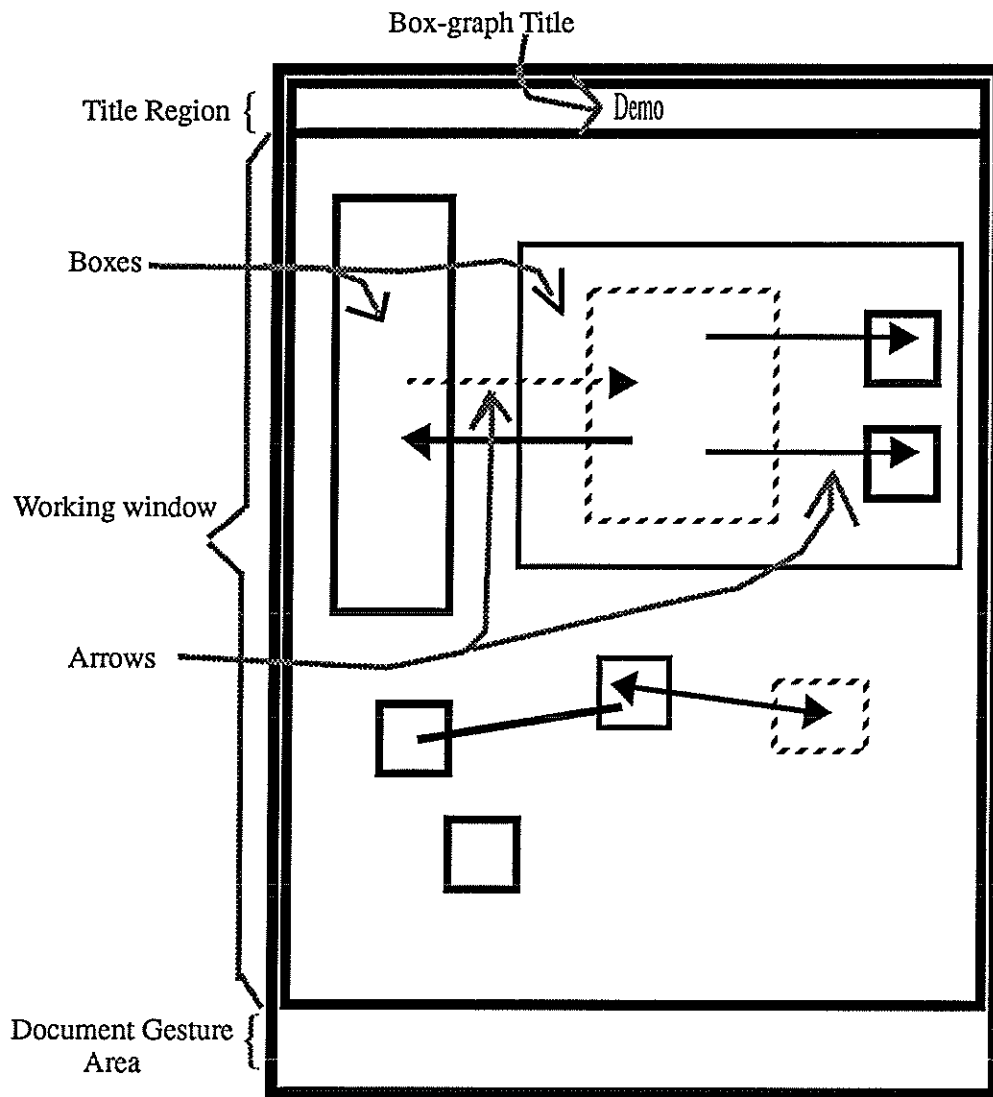


Figure 18. Screen Layout for BAE

5.5. Gesture Recognizer

The implementation of BAE used the algorithm called *Supervised Competitive Learning with Backpropagation Networks(SCL/BP)* [37], as its recognizer for the input gestures. We will discuss the software implementation of SCL/BP, data collection, and the training of the neural network based recognizer. More information on Neural Networks is found in [38]. This is the detailed explanation of the Gesture Recognizer Box in Figure 17.

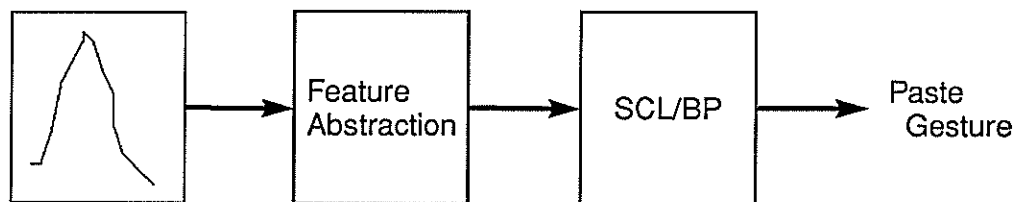


Figure 19. Recognition Process

5.5.1. Recognizer Software

The gesture recognizer, used in BAE was originally developed for character recognition. It consists of two parts; a feature abstraction and SCL/BP classifier. The feature abstraction is used to represent each gesture by a fixed number of attribute values to be used for classification by the neural network. Each gesture is converted into 57 float features, explained below. Figure 20 shows a sample gesture as it is scanned (points) and what the user sees (connected). The 57 features of this gesture are:

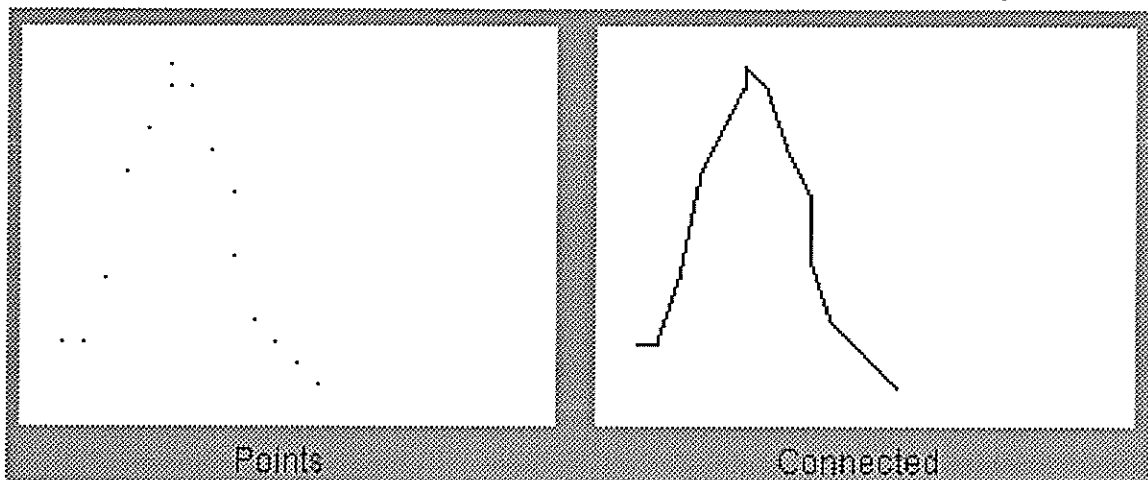


Figure 20. A Sample Gesture (Paste)

Number of Strokes

feature(0)=1.000000

Grid Features

feature(1)=0.00000 feature(2)=0.15000 feature(3)=0.00000 feature(4)=0.00000
 feature(5)=0.00000 feature(6)=0.00000 feature(7)=0.12500 feature(8)=0.12500
 feature(9)=0.00000 feature(10)=0.00000 feature(11)=0.00000 feature(12)=0.00000
 feature(13)=0.00000 feature(14)=0.00000 feature(15)=0.15000 feature(16)=0.00000
 feature(17)=0.00000 feature(18)=0.00000 feature(19)=0.07500 feature(20)=0.00000
 feature(21)=0.00000 feature(22)=0.00000 feature(23)=0.00000 feature(24)=0.00000
 feature(25)=0.02500 feature(26)=0.00000 feature(27)=0.10000 feature(28)=0.00000
 feature(29)=0.00000 feature(30)=0.00000 feature(31)=0.10000 feature(32)=0.00000
 feature(33)=0.00000 feature(34)=0.00000 feature(35)=0.00000 feature(36)=0.15000

Energy Features

feature(37)=0.714286 feature(38)=1.071429 feature(39)=0.714286
 feature(40)=0.714286 feature(41)=1.071429
 feature(42)=0.214286 feature(43)=0.642857 feature(44)=-0.214286
 feature(45)=-0.571429 feature(46)=-0.214286
 feature(47)=0.000000 feature(48)=-0.064103 feature(49)=0.064103
 feature(50)=0.000000 feature(51)=0.000000
 feature(52)=0.320513 feature(53)=-0.256410 feature(54)=-0.256410
 feature(55)=0.000000 feature(56)=0.128205

The features are in three groups, the first being a single number representing the number of strokes of the gesture. The second group, 36 in number, represents the grid features, and the last group of 20 are the energy-related features. The grid features are determined by sectioning the grid into nine sections and within each section looking at the orientations of the gesture stroke. The orientations are vertical, horizontal, positive slope and negative slope (Figure 21 shows the grid and the angles used to determine orientation). The energy features (shown in Figure 22) are the x and y velocities and accelerations, divided into five sections each. A detailed discussion of the feature abstraction algorithm used in the BAE gesture recognizer is given in [48].

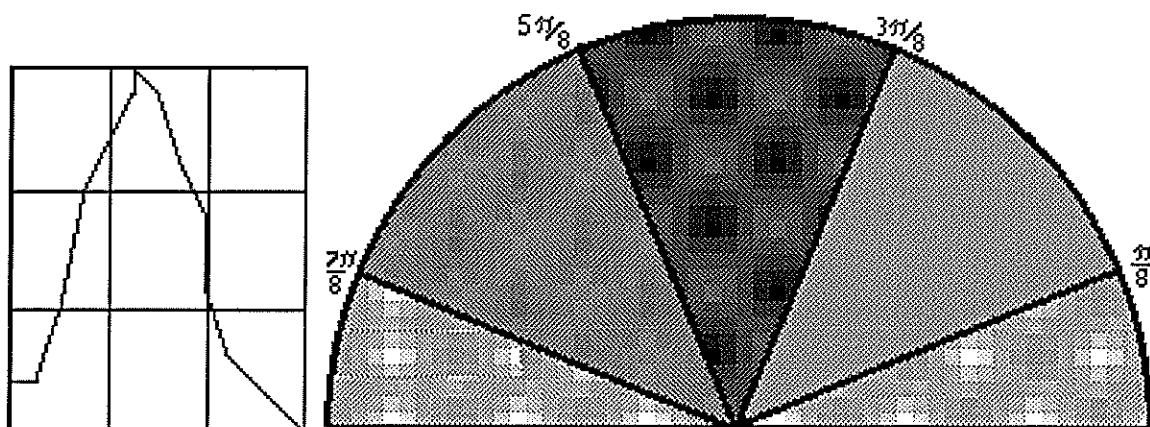


Figure 21. Grid and Orientation Angles

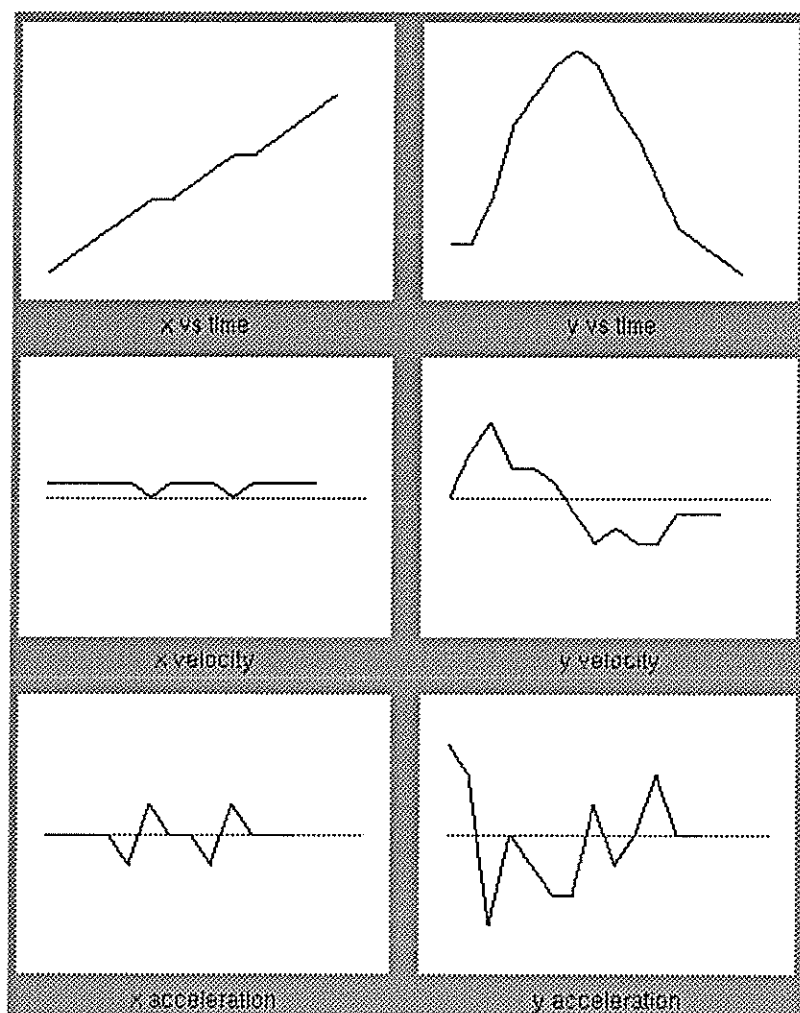


Figure 22. Energy Features (5X4)

SCL/BP is an adaptive learning system that uses Backpropagation Artificial Neural Networks for its learning modules. The SCL system uses a set of prototype units to identify all the inputs of a particular gesture type. Each prototype unit outputs a value that represents the certainty of the input gesture belonging to the gesture type of that prototype. When a gesture is received, it is tested against these prototypes to see if there is a winner, determined by the highest certainty value above a set threshold. If there is no winner then an unused prototype is assigned to the new input, with the correct gesture type assigned to it. If there are no unused prototypes then the prototype with the least frequently used prototype is reassigned to the new input. Then the winning unit is positively trained to achieve a certainty value above a high confidence value, and the losing prototype units (not the same gesture type as the winner), are negatively trained to achieve a certainty value below a low confidence value. An implementation of SCL/BP on the NeXT computer was used as a part of gesture recognition in BAE¹¹.

5.5.2. Data Collection

The gesture data used to train and test the SCL/BP were collected using Lombard with a special data collection software¹². The data is represented as a collection of pen points and each collection is given both a tag number and an id number. The tag numbers are used to separate all the collected gestures, and the id number specifies which gesture type this data belongs to, i.e. cut or help. Both adults and children were used to enter the gesture data. 5460 gestures were collected, representing 34 gestures (three of them were not used in the project - although the system is trained to recognize them). Most of the children were from a Kumon Center, and most of the adults were Kumon Research Group members.

5.5.3. Data Training

The recognizer is trained with the collected data. The system was trained for 34 gestures using 3120 gestures for training and the remaining 2340 for testing. The

¹¹The SCL/BP software was written by Tom Fuller.

¹²The data collection software was written by Dale Frye.

Backpropagation Neural Network has two hidden layers and 57 input features for each gesture. The trained network generated from 80 to 147 prototypes. The model used in BAE has 101 prototypes and was trained until it achieved 85% average accuracy on the testing gestures. This accuracy is average over the 2340 testing gestures and did not reflect the actual accuracy of the system which is discussed in the next chapter.

6. Evaluation

6.1. Gesture Testing and Recognition

The following test was conducted with four adult members of the Kumon Project Group. Participants were shown the BAE system. They were given a sheet containing a list of all the commands available along with their gestures. They were then told to draw a set of predefined objects and operations. This was done to familiarize the users with the system, and help them with understanding the new interface. In the event of misrecognition, the user was told to try again, until the input was successfully recognized. However, the users were told to keep the recognition aspect of the gesture apart when making their judgements on the gesture syntax. The users were then allowed to experiment on their own.

The users were asked to comment on each of the commands - specifying ease of use, ease of remembrance. We also asked for any suggestions to help improve the gestures. We also observed the recognition for the gestures entered.

Table 4 shows the result of this experiment. It shows the gesture, the response of the user in performing the gesture (the syntax), the response of the user in remembering, and the recognition by the recognizer. An analysis of the recognition is given in section 6.6. The participants rated the system using 'easy', 'ok', and 'hard'. In tabulating the results, the gestures that had a mixed response were entered with both the answers. The recognition rating is made from observation. The 'good' rating are given to those gestures that have recognition rate in high 90% range. The 'ok' rating is used for 75% to 95% range and everything below 75% is given a bad rating.

Table 4. Test Results

Gesture	Drawability	Rememberability	Recognition
Open	Easy	Easy	Ok
Save/ SaveAs/ Show Grid	Easy	Easy	Good
New	Easy	Easy	Ok
Close/ Copy	Easy	Easy-Ok	Ok
Cut/ Grid off	Easy	Easy	Bad
Duplicate	Hard	Ok	Good
Paste	Easy	Ok	Good
Grid Panel	Easy	Easy	Good
Align Right	Easy	Easy	Ok
Align Left	Easy	Easy	Ok
Align Top	Easy	Easy	Ok
Align Bottom	Easy	Easy	Ok
Align Vertical	Easy	Easy-Ok	Good
Align Horizontal	Easy	Easy-Ok	Ok
Normal Box	Ok	Easy	Ok
Thick Box	Easy-Ok	Easy-Ok	Good
Dashed Box	Easy-Ok	Ok	Good
Normal Line	Easy	Easy	Bad
Thick Line	Easy	Ok	Good
Dashed Line	Easy	Easy	Good
Normal left Arrow	Easy	Easy-Ok	Good
Normal right Arrow	Easy	Easy-Ok	Good
Thick left Arrow	Ok-Hard	Ok-Hard	Ok
Thick right Arrow	Ok-Hard	Ok-Hard	Good
Dashed left Arrow	Ok-Hard	Ok	Good
Dashed right Arrow	Ok-Hard	Ok	Good
Nor. 2-headed Arrow	Easy	Easy-Ok	Good
Thick 2-headed Arrow	Ok	Ok-Hard	Good
Das. 2-headed Arrow	Ok	Ok-Hard	Good
Help	Easy-Ok	Easy	Good
Quit	Ok	Easy	Good

As seen from the table, most of the gestures were found to be good gestures, at least from the user's point of view. The single gesture found to be bad was the 'duplicate', which was not a simple gesture to draw, and it was not very intuitive, which explains why it did not do well with the users. The other gestures that did not do well with the users were the 'thick' command gestures. At this time, we do not fully understand why. There are two possible explanations. First, users might be responding to these gestures negatively because they are truly not intuitive and not simple to draw. However, they could also be responding negatively, because this particular gesture was not properly introduced. Since the users were given pictures of the gesture with the retrace line not overlapping the original, but drawn slightly above it. An example is the lower left gesture picture in Figure 23 (Thick Box Gesture).

Table 4 might indicate *on average* a high gesture recognition rate. However, this does not reflect the real usage of the gestures. Although the recognizer was trained to achieve 85% accuracy, the users were very frustrated with the recognition, because the two most frequently used gestures, the 'simple' line and the 'delete' gesture were not successfully recognized. A better measurement of recognition rate would be to use weighted average based on frequency.

6.2. Data Collection

Some of the gesture data used to train the Backpropagation Neural Networks were collected from children. Ten different gestures were collected from twenty three students between the ages of eight and fourteen. See Figure 23 for the gestures asked to be drawn by the children. Figures 24,25, and 26 show some of the gestures drawn by the children.

The most serious problem noted during the collection was the unfamiliarity of the children using a special pen and writing on a glassy surface. Similar problems were reported on earlier experiments with children [7]. The other problem, in collecting data from children, deals with children younger than eight years. Their attention span was

very short and their careful attention to detail might have been their own undoing, since this caused them to be too careful and too slow leading to a faster loss of attention and more mistakes. The recognizer as it stands today tries to gather as much information on the scribbles as possible in trying to identify it. This however causes problems when children in their attempts to make sure everything looks okay, tend to touch up their drawings, leading to multiple stroke gestures when the recognizer is looking for a single stroke. An example of this is Figure 25, the 'paste' gesture is 'touched up' to look just like the example picture. A combination of the previous two problems can be seen in Figure 26. The child entered the first half of the gesture, forgot what the shape looked like so stopped to look at the picture, then came back and added the second stroke to finish the gesture.

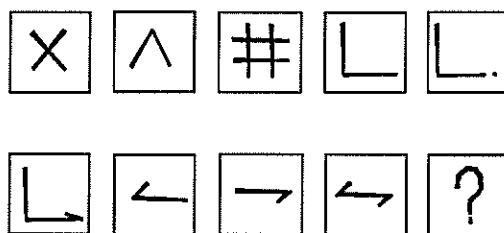


Figure 23. Gestures Asked to be drawn by Children

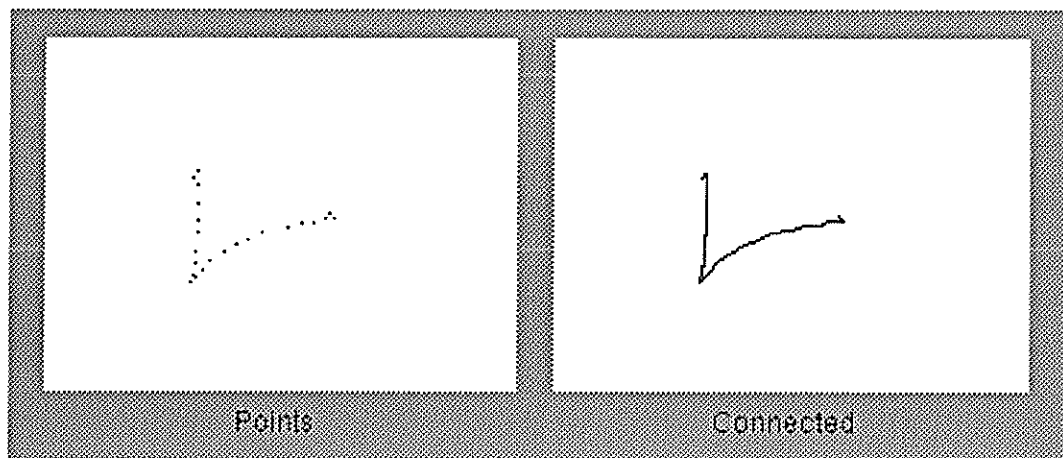


Figure 24. The 'box' gesture entered by a Child

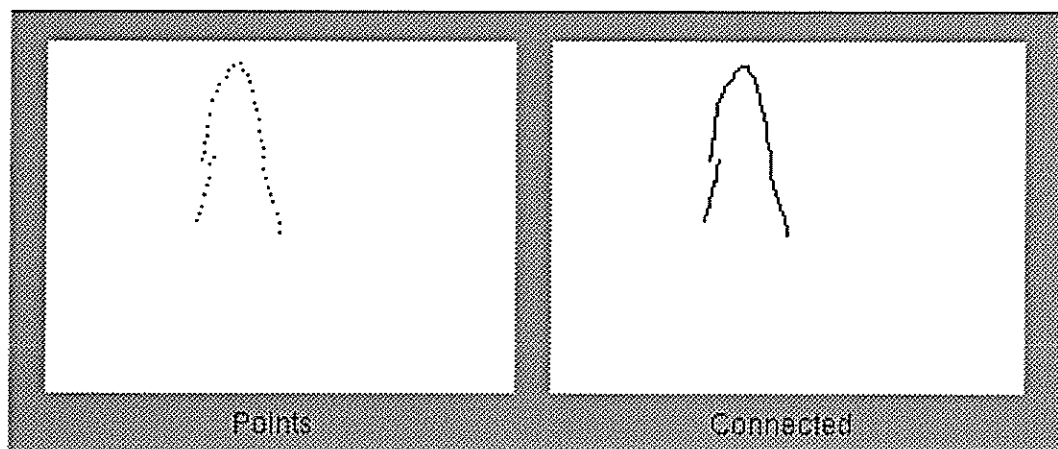


Figure 25. The 'paste' gesture entered by a Child

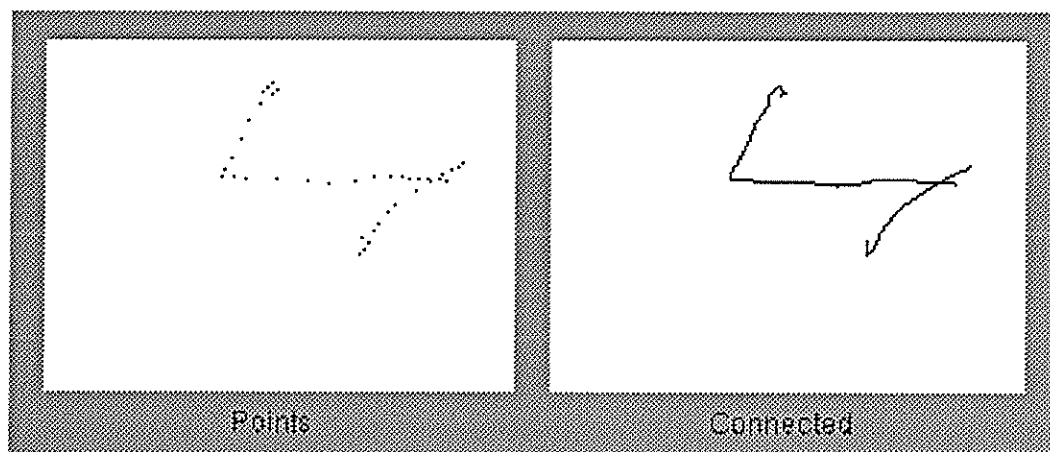


Figure 26. The '2 Headed Arrow' (normal) gesture inputted by a Child

6.3. Implementation

The implementation still has some problems to be worked out, but as far as the ability to test the gestures, it works fine. However, for a better evaluation of the gestures more semantic information needed to be added to the editor. The evaluation of such a system would be a more appropriate test-bed for the Hyperflow gesture system design.

6.4. Choice of Test-bed

The choice of the test-bed for this project was limited. The problems with the NeXT tablet interface, which was not yet completed, did not allow us to use a tablet for the input/output device. The Kumon Machine, a pen computer for children, which would have been the ideal choice, was not yet available. And the lack of knowledge of the PenPoint Operating system and how to develop programs with it, ruled out the possibility of developing BAE on the Lombard alone.




The only solution that allowed the least amount of training time, was to program in C on the NeXT computer and have simple primitives on the Lombard to support the BAE system. This is the implementation that is described in this thesis.





6.5. Use of Menus

Although we set out to design a gesture system such that gestures were the only means of interaction between the user and the computer, this was not completely satisfied. Some of the commands, like Open and SaveAs, uses a menu in this implementation. We decided to concentrate on the first level of editor's commands as the target of our gesture design, for manageability. We simply used menus and window boxes to do the remaining commands. In a fully gesture driven system, this would not be acceptable. For example, the boxes used to input the alphabet and numbers would be replaced with a character recognizer.

6.6. Observation

(1) The recognition system, imported from the character recognition application, was one of the drawbacks. The number of prototypes needed to successfully identify such closely linked gestures made it difficult to get good recognition results. It is important to learn the capabilities and incapacities of the recognizer chosen to use. In BAE's

case the recognizer really needed to have distinct differences between the gestures, any similarity would cause problems. Such as the 'line' gesture, , the 'dashed line' gesture, , and the 'thick line' gesture, . In this case, the 'dashed line' gesture and the thick line' gesture were correctly recognized a great majority of the time, but the line gesture's recognition rate was in the 60%-70% range. The system had a very high recognition (almost 100%) when dealing with complex gestures (this with the added premise that the gestures were drawn correctly in the first place!!). This is a direct result of the feature abstraction, originally designed for character recognition. The 'dashed line' and the 'thick line' gestures have more information to clearly distinguish them. A similar recognition problem was seen with the combinations of the

'thick left' arrow gesture  (top) with the 'normal 2-headed arrow' gesture  and the 'thick line' gesture  with the 'normal right arrow' gesture  (bottom). These gestures are very similar to each other, the angle of the ending segment is very important. If the angle is clear, then it is a normal type with a right arrow. If there is no angle, i.e. its a retrace over the last section, then this is specifying the 'thick' type on the arrow.

In choosing the SCL/BP, which was designed for character recognition, we believed that we would overcome the recognition problems with the addition of tap detector and hot point detector. The results of the testing, however, showed that this approach did not work. The feature abstraction used for characters did not properly work for gestures. The next recognizer needs to have feature abstraction that is specifically designed for recognizing gestures. These results do not necessarily rule out the use of Neural Network for gesture recognition, however the feature abstraction needs to be changed.

(2) The second issue was the simplicity of a gesture. We had to differentiate between the three types of the same box and arrow. Our solution was to change the ending of each gesture such that the ending specified the type. This turned out to be a correct

decision from the results of the box gesture. The arrows were more complex since they also had direction information, i.e. the arrow heads. When we combined both the different kinds and different types of arrows and tried to create gestures to deal with both at the same time, the gestures became too hard to perform and too hard to remember. There was also no connection between any gesture, every part was fixed, so if the gesture was recognized wrongly either the kind of arrow or the type, the whole gesture had to be deleted and started over again, until the recognizer correctly recognized. This showed that there has to be a method of changing the type of each object once it has been created on the screen, saving time on attempts to delete and create again. This also would have had its drawbacks in the sense that there would have been more gestures to perform to obtain the object, or more gestures to remember to correct the mistakes.

(3) Last issue is the use of overloading. Three of the gestures were overloaded, 'copy'/'close', 'save'/'saveas'/'show grid' and 'cut'/'turn off grid'/'multiple cut'. This had mixed results. The overloading of the save and saveas were very intuitive and proved not to be a problem, but show grid 's' was not a very intuitive answer. So it was not a problem of overloading but just a bad initial design of having unintuitive gestures for commands.

6.7. Design Process.

In this section I discuss the evaluation of the design principles: simplicity, drawability, complexity, rememberability, and intuitiveness. Of all these complexity was the least involved since none of the gestures had a hot point placement that was difficult to remember or perform. The simplicity of using single stroke gestures was not fully tested in this thesis, although I believe it would have strong effects in recognition and rememberability. Drawability was also not much of a factor, since almost all gestures were simple one and two stroke gestures. Rememberability has a lot to do with intuitiveness, simplicity and experience. The more the users used the gestures, the

easier it was to remember them. The simpler gestures were remembered faster, and those gestures that were **not** intuitive, no matter how simple, were not easily remembered. The major issue was intuitiveness. As I found out, what was intuitive to me was not at all intuitive to others.

In terms of the process, these design principles are very good place to start in designing the gestures, but a thorough user testing is a must to deal with the subtle differences that arise in the process.

7. Conclusion

In this thesis we developed a test-bed application to gain insight into designing gesture systems for pen based computers. This work is a preliminary study for gesture systems being designed for the visual programming language Hyperflow, and as a user interface tool for the Kumon Machine.

This study confirmed the usefulness of the requirements for gesture design, and in conclusion stressed the importance of careful study and incorporation of recognizer strengths and weaknesses in the design process of gestures.

The results of the recognition showed that more needs to be done in helping the user deal with misrecognitions. An 'undo' command is a must, and maybe a user controlled second guess at times of misrecognition will help with speedy recovery from misrecognitions.

Problems in dealing with the retrace concept, leads us to conclude that a two dimensional representation of the gestures might not be the correct way to introduce the gestures to the users. A better alternative would be to have the system demo the drawing of the gestures to the users. This way the user will see the time component and will actually get a better visual input for each gesture.

The next step, in terms of a recognizer, should be to test the recognition capabilities of SCL/FZ, mentioned in chapter 1. With the results it is obtaining with character and digit recognition, it might have a better recognition rate than SCL/BP.

For the test-bed, it should be on a single platform. Either it should be converted to be on the Lombard or NeXT with a tablet attachment. This would allow the use of more complex drawing primitives to help with the speed up of drawing.

For the BAE system itself, the Editor should incorporate more semantics. To be a better test-bed for Hyperflow gesture design, the editor should incorporate some of the semantics associated with the boxes and arrows in the Hyperflow design.

8. Acknowledgements

I would like to thank Dr. Takayuki Dan Kimura for his comments and help in writing this thesis. I would like to thank Tom Fuller for providing SCL/BP recognizer, Dale Frye for writing the draw primitives on the Lombard system and Ed Hutchins for his coding advise and List code. I would also like to thank the many contributors of Graphics Gem and Graphics Gem II for their very helpful code.

I also would like to thank Dr. Kenneth Goldman and Dr. Anne Johnstone for their comments and suggestions.

Figure 2 Printed with permission from William Buxton and Gordon Kurtenbach

Figure 3 Printed with permission from Dean Rubine.

Figure 5 Printed with permission from GO Corporation.

9. BIBLIOGRAPHY

- [1] Davis, M. R. and Ellis, T. O., "The Rand Tablet: A Man-Machine Graphical Communication Device", AFIPS Conference Proceedings 1964 FJCC, no 26, pp 325-331, 1964.
- [2] Teixeira, J.K. and Sallen, R.P., "The Sylvania data tablet: a new approach to graphic data input.", Proceedings of the Spring Joint Computer Conference, pp 315-321, 1968.
- [3] Wolf, C. G., Rhyne, J. R. and Ellozy, H. A., "The Paper-Like Interface," Designing and Using Human-Computer Interfaces and Knowledge Based Systems, IBM T. J. Watson Research Center, Yorktown Heights NY 1989 pp 494-501.
- [4] Kimura, T. D., "Silicon Paper and A Visual Interface for Neural Networks," Proceedings of 1990 IEEE Workshop on Visual Languages, Chicago, IL October 1990, pp 241-246.
- [5] Rhyne, James R. and Wolf, Catherine G., "Gestural Interface for Information Processing Applications", Research Report 12179 (#54544) 9/2/86, T.J.Watson Research Center, IBM Corporation, P.O.Box 218, Yorktown Heights, NY 10598, September 29th, 1986
- [6] Wolf, C. G., "Can people use gesture commands?", SIGCHI Bulletin, 18, pp 73-74. Also IBM Research Report 11867.
- [7] Shepard, S. R., "Gesture Analysis for the manipulation of graphic objects", Technical Report NSF/ISI-87113, Sensor Frame Incorporated, 4516 Henry Street, Suite 505, Pittsburgh, PA 15213, Sept. 1987.

- [8] Wolf, Catherine G. and Morrel-Samuels, Palmer, "The use of hand-drawn gesture for text editing", *Int. J. Man. Machine Studies*, Vol. 27, pp 91-102, 1987.
- [9] Shneiderman, B., "The future of interactive systems and the emergence of direct manipulation", *Behavior and Information Technology*, 1, pp 237-256, 1982.
- [10] Wolf, Catherine G. and Rhyne, James R., "A Taxonomic Approach to Understanding Direct Manipulation", Research Report 13104 (#58210) 9/4/87, T. J. Watson Research Center, IBM Corporation, Yorktown Heights, N.Y. 10598, September 1987.
- [11] Wolf, Catherine G., "A Comparative Study of Gestural and Keyboard Interfaces", Research Report 13906 (#62477) 8/5/88, T.J. Watson Research Center, IBM Corporation, Yorktown Heights, N.Y. 10598, August 1988.
- [12] Kimura, T.D., Choi, J.W. and Mack, J.M., "A Visual Language for Keyboardless Programming," Technical Report WUCS-86-6, Department of Computer Science, Washington University, ST. Louis, MO, June 1986.
- [13] Rhyne, Jim, "Dialogue Management for Gestural Interfaces", *Computer Graphics*, Vol. 21, No.2, April 1987, pp. 137-142. Also IBM Research Report RC 12244, 1986.
- [14] Tappert, C.C., "Cursive Script Recognition by Elastic Matching", *IBM J. Res. Develop.* Vol. 26, No. 6, November 1982
- [15] Kao, H.S., Van Galen, G. and Hoosain, R., eds., "An Adaptive System for Handwriting Recognition", *Graphonomics: Contemporary Research in Handwriting*, Elsevier Science Publishers B.V. (North-Holland) 1986.

[16] Tappert, C.C., "Speed, Accuracy, Flexibility Trade-Offs in On-Line Character Recognition", Research Report 13228 (#59158) 10/28/87, IBM Research Center, T. J. Watson Research Center, Yorktown Heights, N. Y. 10598, 1987.

[17] Kankaanpaa, A., "FIDS - A Flat Panel Interactive Display system", IEEE Computer Graph and Appl., No. 8, p 71-82, March 1988.

[18] Tappert, C., Fox, A., Kim, J., Levy, S. and Zimmerman, L., "Handwriting Recognition on Transparent Tablet over Flat Display", Proceedings of SID, Vol 28, No. 1, 1987.

[19] Fujisaki, T., T. E. Chefalas, J. Kim and C. C. Tappert, "Online Recognizer for Runon Handprinted Characters", Proceedings of the tenth International Conference on Pattern Recognition, Atlantic City, New Jersey, June 16-21, 1990. pp 450 -454.

[20] Tappert, C. C., "Online Character Recognition by Decomposition Matching", Research Report RC 15478(#68849) 2/9/90, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, 2/9/90.

[21] Wolf, Catherine, G., "Understanding Handwriting Recognition from the User's Perspective", Research Report RC 15958(#70552) 6/21/90, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, June 1990.

[22] Tappert, Charles C., Ching Y. Suen and Toru Wakahara, "The State of the Art in On-line handwriting Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August 1990, pp.787-808.

[23] Kim, Joonki, "On-line Gesture Recognition by Feature Analysis", Proceedings of Vision Interface '88, June 6-10, 1988, pp 51-55.

[24] Burns, L. M., D. Orth, S. Perkins and H. Ginsburg, "Meadow: Mathematical Errors and Automatic Debugging of Written Input", 10/31/1990.

[25] Martin, Gale L. and Pittman, James A., "Recognizing Hand-Printed Letters and Digits Using Backpropagation Learning", *Neural Computation*, No. 2, 1991, pp. 258-267.

[26] Chow, Doris, and Kim, Joonki, "Paper Like Interface for Educational Applications", National Educational Computing Conference, Boston, Massachusetts, June 20-22, 1989, p 337-344.

[27] Andreshak, J. C., S. Lumelsky, I. F. Chang, T. P. Mears, A. A. Stone and W. W. Stead, "Medication Charting via Computer Gesture Recognition", Research Report RC 16024 (#69114) 3/5/90, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 10598, 3/5/90.

[28] Henry, T.R., Hudson, S.E. and Newell, G.L., "Integrating gesture and snapping into a user interface toolkit", in *UIST '90*, pp 112-122, ACM, 1990.

[29] Buxton, W. A. S., and Kurtenback, G., "Editing by contiguous gestures: a toy test bed", *ACM CHI '87 Poster*, 1-5, 1987.

[30] Kurtenback, Gordon and Buxton, William, "Issues in combining marking and direct manipulation techniques", *UIST '91*, pp 137-144, ACM, 1991.

[31] Lipscomb, James S., "A Trainable Gesture Recognizer", *Pattern Recognition*, Vol. 24, No. 9, 1991, pp. 895-907.

[32] Rubine, Dean, "Specifying Gestures by Example", Technical Report, Information Technology Center, Carnegie Mellon University, Pittsburgh, PA, 1991.

- [33] Rubine, Dean, "Integrating Gesture Recognition and Direct Manipulation", Usenix Technical Proceedings. summer 1991.
- [34] Kimura, T. D., "Hyperflow: A Visual Programming Language for Pen Computers", submitted to 1992 IEEE Workshop on Visual Languages.
- [35] Selker, T., Wolf, C. and Koved, L., "A Framework for comparing systems with visual interfaces", Human-Computer Interaction- INTERACT '87, Elsevier Science Publishers B.V. (North-Holland), 1987.
- [36] Raeder, G., "A survey of current graphical programming Techniques", I.E.E.E. Computer, Vol. 18, No. 8, August 1985, pp 11-25.
- [37] Kimura, T. D. and Fuller, T. H., "Supervised Competitive Learning Part I: SCL with Backpropagation Networks", submitted to ANNIE'92 (Artificial Neural Networks in Engineering).
- [38] Schalkoff, Robert, Pattern Recognition: Statistical, Structural and Neural Approaches, New York: John Wiley & Sons, Inc., 1992, Ch. 1, Ch. 10-13.
- [39] Kurtenback, G. and Buxton, W., "GEdit: A test bed for editing by contiguous gestures.", SIGCHI Bulletin, Vol. 23, No. 2, pp 22-26, ACM, 1991.
- [40] Carr, R. and Shafer, D., The Power of Penpoint, Massachusetts: Addison-Wesley, 1991.
- [41] Pen Windows, Microsoft Pen Windows Programmers Manual, Microsoft, 1991.
- [42] Draw, Draw Application Program by Steve 1987.

- [43] TopDraw, TopDraw User Manual , TopDraw Version 1.0d, Media Logic Incorporated, 1990.
- [44] Bartow, T. S., "Gesture: Present Systems and Philosophy", Kumon Project Memo, Washington University, July 1,1991.
- [45] Kimura, T.D., "Learning Math with Silicon Paper," Technical Report WUCS-92-12, Department of Computer Science,Washington University, St. Louis, MO February 1992.
- [46] Grossberg, S. "Competitive Learning: From Interactive Activation to Adaptive Resonance", Cognitive Science, No 11. pp. 23-63, 1986.
- [47] Kimura, T. D. and Wang, C., "Supervised Competitive Learning Part II: SCL with Fuzzy Logic", submitted to ANNIE'92 (Artificial Neural Networks in Engineering).
- [48] Fuller, T. Jr., "Energy-related Feature Abstraction for Handwritten Digit Recognition", submitted to Fourth Midwest Artificial Intelligence and Cognitive Science Conference, May 1992.

Burak M. Taysi

VITA

Date of Birth: 3/20/68

Place of Birth: Philadelphia, Pennsylvania

Undergraduate Study: Washington University,
St.Louis, Missouri
B.S., 1992

Graduate Study: Washington University,
St. Louis, Missouri, 1990-Present
M.S., 1992

Professional Societies: A.C.M., I.E.E.E.

Honors/Awards: Eta Kappa Nu
CS Department Service Award 1991

Professional Experience: Consultant/Grader, Washington University,
1990-1992

August 1992