Chapter 9:

SOME REPRESENTATIVE 2D TASKS

Introduction

One way to try and understand these issues is to experiment with a diverse set of representative tasks. Each task has its own idiosyncratic demands. Having enumerated such a set, we can determine which properties are relevant to a particular application, and then test the effectiveness of various technologies in their performance. This allows us to get a rapid match of technology to the application. Furthermore, the set of representative tasks provides a reminder of what dimensions should be considered in the selection process.

In the remainder of this section, we describe a basic set of such generic transactions through a set of small test programs. These make an excellent implementation project for the student, and provide a good test-bed for developing a strong understanding of the characteristics of several input devices.

Like any other list, this one is not complete. It focuses on 2D tasks, such as those found in many graphical user interfaces. Text entry and 3D input, for example, are not emphasized. We leave it as an exercise to the reader to expand on the list.

Pursuit Tracking:
Tracking is one of the more studied tasks in HCI. The classic text on the subject is Poulton, E.C. (1974). In this section, we describe a simple 2D tracking task.

In this test, a moving target (a fly) moves over the screen under computer control. The operator uses the control device to track the fly's motion. Feedback about the operator's performance is given by a tracking symbol in the form of a fly swatter. The idea is to see how many times the fly can be killed by positioning the swatter over the fly and pushing a button device.

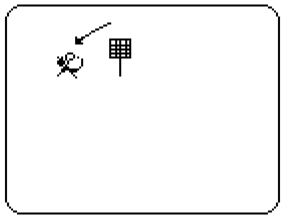


Figure 6: Pursuit Tracking
A moving target (a fly) moves over the screen. The tracking symbol is a fly swatter. When the swatter is over the fly, the user pushes the button to "swat" the fly.

The main statistic in this test is how many times the fly can be swatted in a given time interval. There are a number of parameters that you want to have variable so that you can develop an understanding of their influence on the task performance. One of these is the speed that the target moves. Another is the *control:display (C:D) ratio*.

One other consideration is the button push that is required to swat the fly. For example, can this be activated with the same hand that is providing the spatial control? This may be difficult if a touch-tablet or track-ball is used. If the same limb providing the spatial control cannot be used for the button event, is another limb free in the application being tested for?

Target Acquisition:
In this task, the user replicates Fitts' reciprocal tapping task, as illustrated in Fig. 2. Only one pair of bars is used at a time. In a given interval of time, the user is to tap back and forth between bars as many times as possible. Moving to a bar and tapping is intended to be an abstraction of actions such as selecting icons.

Examine how target size and target distance affect the speed of target acquisition. The smaller the target the longer it will take to acquire, due to the fine motor control required. Verify if your data conforms to Fitts' prediction. Use different devices.

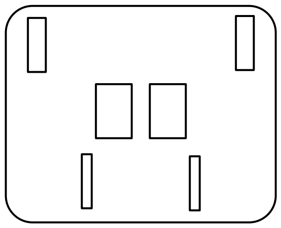


Figure 2: Target Acquisition
Three representative Fitts-type reciprocal tapping tasks. For each pair, measure how many times you can tap back and forth between each within a given time interval. Use different bar widths and distances, as suggested by the figure. Compare the performance of different devices..

As in the pursuit-tracking task, issues such as C:D ratio and what button device is used will affect performance.

There are a number of variations on this basic task. Each brings to light important aspects of the input technology:

Homing Time: If an application involves frequently moving between a text entry device (usually a QWERTY keyboard) and a pointing device, it is important to know the effect of this movement on performance. This is the *homing time* discussed in the Keystroke Model (Card, Moran & Newell, 1980). To get a feeling for the effect of homing time, have the user push a key on the keyboard (the space-bar, for example), after each square is selected. Using the same tablet, for example, what is the difference in performing this task when using a puck *vs* a stylus?

Testing State-2 Target Acquisition: Dragging and Rubber-Band Lines: In this variation, a rubber-band line is stretched between bars. Tapping anchors one end of the line. The button must be held down until during motion to the opposite bar. Releasing the button anchors the second end. How does performing the task in State-2 compare to State-1?

Free-Hand Inking:
Attempting to input a facsimile of your handwritten signature places yet another set of demands on the input technology. To get a feeling for the degree to which various devices lend themselves to this type of task, present the user with a screen ruled with lines of decreasing spacing. Have the user sign their name once in each space, as illustrated in Fig. 3. Use a simple subjective evaluation of the quality of their signature as a means of comparing devices. The attributes of this hand-writing task are relevant to several other common tasks, such as those seen in drawing programs, gesture-based interfaces, and character recognition systems.

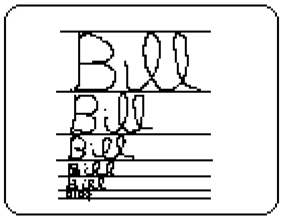


Figure 3: Free-Hand Inking
A device's ability to capture a good facsimile of your signature is a good measure of its effectiveness for inking and drawing tasks. Comparisons can be made among signatures of different sizes to better understand the effect.

Note (as a place holder until the appropriate chapter is written): There are a number of papers that describe novel and techniques that greatly expand upon how inking is normally done. To investigate these techniques, see the following references:

- Bleser, Sibert & McGee (1988)
- Green, R. (1985)
- Johnstone, E. (1985)
- Lee, S.K., Buxton, W. & Smith, K.C. (1985)
- Ware, C. & Baxter, C. (1989)

Tracing and Digitizing:
In many applications, such as cartography, CAD, and the graphic arts, it is often important to be able to trace material previously drawn on paper, or digitize points from a map. There is a wide variation in how well various devices can perform this type of task. Relative devices such as mice and trackballs are almost useless in this regard. Even with absolute devices like tablets, for example, there is a wide variation in their ability to perform this class of task. In particular, the demands on resolution and linearity vary greatly across applications. In cartography and engineering, for example, the accuracy of the digitization is often critical and far beyond what is required in digitizing a sketch.

Constrained Linear Motion:
In some applications it is important to be able to move the tracker rapidly along a straight-line path. One example is in using the scroll-bar mechanism of some systems. Another example might be where you want to use the motion of a mouse in one dimension, say Y, to control one

parameter, without changing the value of another parameter being controlled by motion in the other dimension, X. This is similar to the problem already encountered in using the Skedoodle's joystick to draw a square. We discovered its importance in studying virtual devices for the study by Chen, Mountford and Sellen (1988).

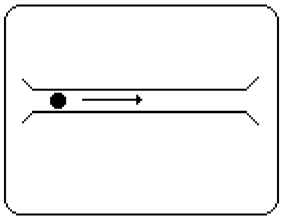


Figure 4: Constrained Linear Motion
How quickly can the ball be moved along the straight path without crossing the lines?

Different devices vary in the ease with which this can be done. X/Y thumb-wheels, for example, would out-perform a mouse in the above task *if* the motion was along the primary axes. In the example task, illustrated in Fig. 4, the tracking symbol is a ball that is dragged along a linear path defined by two parallel lines. The object is to move along the path without crossing the parallel lines. How is speed affected by the input device used and the path width? Are similar results obtained if the path is vertical or diagonal?

Constrained Circular Motion:
A variation of the previous example is to see how well the user can specify a circular motion within a constrained region. This type of control is useful in manipulating 3D objects, for example, and is described in Evans, Tanner and Wein (1981) and Chen, Mountford & Sellen (1988). As in the previous example, different results will be obtained from different devices, and results will also vary according to C:D ratio and the width of the path.

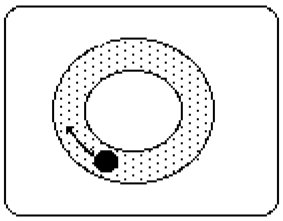


Figure 5: Constrained Circular Motion
How fast can the tracking symbol be moved around the circular shaded path without crossing the borders?

Compound Tasks

AN INFORMAL STUDY OF SELECTION-POSITIONING TASKS[1]

[1] Based on the original version published in Proceedings of Graphics Interface '82, Toronto, May 1982, 323 - 345.

INTRODUCTION

In this section, several techniques for performing selection-positioning tasks are compared. The comparison takes the form of a case study. The task, selection-positioning, is chosen because it is a *compound* task, unlike the simpler tasks examined in Chapter 1. As we proceed, we will see that how simple tasks *chunk* together in such compound units is important to understand and control.

The task is to select from among three geometric shapes and position them in two-space. The study emphasizes how much syntactic, lexical and pragmatic variables can influence the relative ease with which a particular task can be performed. It is shown how each approach has properties which make it optimal in some contexts. The overall impact of the study is to demonstrate the importance of actually testing ideas and to point out the shortcomings of pencil and paper exercises.

The work is rooted in the observation that systems tend to be constructed out of sets of reoccurring classes of transactions, regardless of application. For example, whereas a musician might select a note and place it in a particular position in pitch and time, so might a circuit designer place a particular gate in its proper place in a circuit. While what is accomplished in each case is quite different, both constitute the same generic transaction: a selection-positioning task.

This observation is not new and forms the basis for graphics "logical devices" (GSPC, 1979; ACMCS, 1978). The problem is that the result is not of a fine enough grain of analysis. The graphics categorization does not penetrate much below the syntactic level. The logical devices let us describe our musician's and circuit designer's actions in terms of "picking devices" and "locators", but do not contribute much in the way of characterizing the properties of the picking devices used, for example.

Foley and Van Dam (1982) describe the user interface as consisting of four layers:

- conceptual
- semantic
- syntactic
- lexical

To this list we would add:

- pragmatic

Our point is that designing with logical devices is useful, but incomplete. This is due to an inability to adequately deal with the lexical and pragmatic levels.

CASE STUDY OVERVIEW

If one step of systems analysis is to categorize a particular transaction, then the next step is to understand the alternatives within that tcategory. Alternatives at this level are distinguished by syntactic, lexical and pragmatic properties. It is these properties which we want to investigate. To do so, we implemented an environment which supported several different ways of performing one simple task: selecting a shape from among a square, circle and triangle, and positioning it in two dimensional space.

To provide a common basis for comparison, the techniques implemented used the same hardware: a graphics tablet, a vector-drawing display and an ASCII keyboard.

Five basic techniques for performing the designated task were implemented:

- Dragging
- Moving-Menu/Stationary-Pointer
- Character Recognition
- Typing
- Function Keys

None of the techniques (except perhaps the second) is original. The value of the exercise lies in implementing the techniques in a common environment and comparing them in actual use.

DRAGGING

Simple Case

The first technique, "dragging", is illustrated in Fig. 1. The user selects the desired shape by positioning the tracking cross over the appropriate menu item (on the right side of the display), and depressing the tablet cursor's selector ('Z') button.

A copy of the shape is thereby "picked-up", and follows the cursor's motion (for as long as the Z-button remains depressed). The shape can be positioned, therefore, by cursor movement and "anchored" by releasing the Z-button when in place.

The technique is simple and well known. Nevertheless, it brings to light a few interesting questions. For example, how does the location of the menu region affect the interaction? When, if ever, is it an advantage to have menus positioned along the top or bottom of the screen?

A less obvious issue is seen in the interplay between the syntactic and lexical components of the example. There are at least two ways to perform the task. A Z-down/drag/Z-up gesture, for example, binds the constituent operations into a single connected gesture.

This is in marked contrast with the common alternative of a Z-down/Z-up move Z-down/Z-up command sequence. In this latter case, the first Z-up is an act of closure which could disrupt the binding of the sub-tasks, and the coherence of the overall task performance.

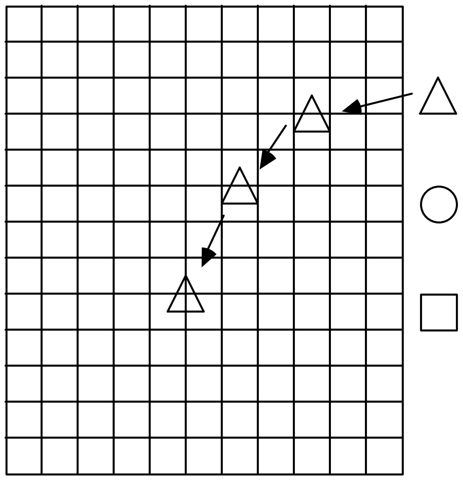


Figure 1: Dragging

Redundant Case

In the previous version of the dragging technique, the user had to go back to the menu to select an item before each placement. If not, depressing the Z-button resulted in the tracking symbol becoming a question mark icon. This is clearly inefficient in cases where several instances of the same shape are to be laid down.

A variation on the technique is to use a "paint-pot" analogy and have the last item selected "remain on the brush". Once the first instance of a shape has been selected and positioned, subsequent instances of that shape can be input by a simple Z-down/Z-up gesture at the desired position.

While this modification will often result in a more efficient system (as measured by the keystroke model of Card, Moran & Newell, 1980), the potential savings will not always be taken advantage of by the user.

For example, a designer will not generally lay out all the AND gates and then all the OR gates of a circuit. Rather, the circuit may be built up in logical order. The semantics of the task will dictate the order rather than syntactic/lexical efficiency.

Recognition of such facts will help the interface designer to prioritize where effort should be invested in attempting to improve the quality of the user dialogue.

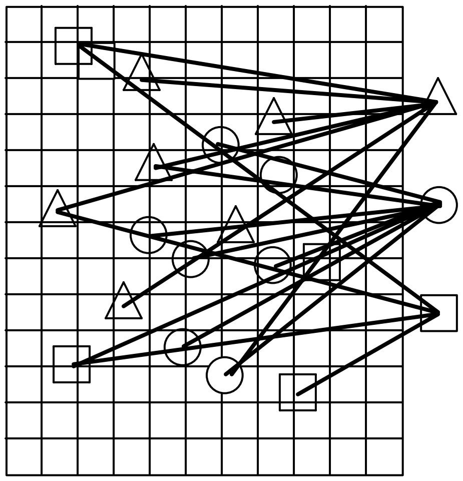
MOVING-MENU/STATIONARY-CURSOR

Simple Case

With the dragging technique, much hand motion was expended in moving between the menu and the work areas.

This is illustrated in Fig. 2, where we have drawn vectors to connect all points on the screen where an interaction occurred during the course of performing a simple layout task[2].

One way to save much of this hand motion (and the time that it consumes) is to have the menu come to us rather than us going to it.



[2] Diagrams such as this can be made from the "dribble file" in which a time-stamped record of all interactions can be placed. The data that results can be used in the evaluation of the user interface through the technique of protocol analysis. Such a file can be generated on request from any program which uses our menu-support tools. The example hints at the potential value of such tools, but also forces us to realize the inadequacy of our current knowledge concerning techniques of protocol analysis, *viz.*, how to interpret the data.

Figure 2: Hand Motion in Dragging

The technique implemented provides many of the same advantages as a pop-up menu on a raster display. We place the tracking cross over the position where we want a shape and depress the Z-button. This causes two things to happen (for as long as the Z-button remains depressed):

- the tracking cross becomes anchored at its current location.

- the menu seen in Figure 3, becomes the tracking symbol *i.e.*, it follows the motion of the tablet's cursor.

Conceptually, what we now have is a moving menu and a stationary pointer. The shape desired is selected by placing it over the (stationary) tracking cross and releasing the Z-button.

The technique is effective in many contexts. However, it has some interesting properties and begs some important questions. When compared to dragging, for example, it breaks down as the number of items on the menu increases.

Also, the technique is not self-documenting. That is, there are no explicit cues to prompt the user as to the nature of the interaction. Dragging, on the other hand, may be considered more obvious. On the other hand, the selection menu of the new technique does not consume any permanent display real-estate, thereby providing a larger effective work area.

Notice that the syntax of the transaction is the reverse of dragging, where items were first selected, then positioned. There is some question as to whether it is more "natural" to perform the selection task first. The question is important in its own right, but there is also a more global issue.

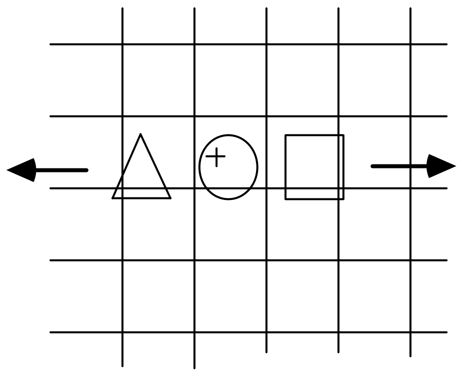


Figure 3: Moving Menu / Stationary Pointer

Barnard, Hammond, Morton, Long and Clark (1981) give evidence as to the importance of self-consistency of syntax within a system. If, for example, there are several compound tasks that involve selection, the evidence suggests that the selection task should appear in the same syntactic position in each case. The literature is not conclusive and a great deal of work remains to be done. In the meantime, however, these considerations should be kept in mind by designers.

Redundant Case

The moving-menu/stationary pointer technique can also be implemented to facilitate entering several instances of the same shape. This is accomplished by, on the Z-down, always having the menu appear positioned such that the last selected shape appears over the anchored pointer. Thus, for the second instance on, the input gesture is exactly the same as that for the second instance on in the redundant mode of dragging: a simple Z-down/Z-up.

SHORTHAND RECOGNITION
The shorthand recognition technique integrates the selection and positioning tasks into a single gesture. To input a shape, a short-hand symbol is sketched at the desired location in the work area. The shorthand symbols used in our example are shown in Fig. 4.

Like the moving-menu/stationary-pointer technique, this approach requires no display real-estate for control functions. However, it also shares the property of not being self-prompting.

One of the main issues of shorthand recognition approaches has to do with the cognitive burden of remembering the symbols[3].

One approach is to use a trainable recognizer. This may be based on the premise that it is easier for users to remember symbols which they have designed themselves. Whether this is true or not depends on several factors, including the number of characters, their complexity and what they represent. The training process imposes a cognitive (and temporal) burden of its own, and such character recognizers usually respond more slowly that ones which operate on a predefined set of symbols.

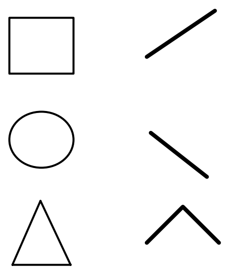


Figure 4: Shorthand Symbols

One way which the short-hand symbols can be made easier to remember is to make them iconic. Thus symbols might look like transistors, AND gates, or squares, for example. There are problems with this, however. As the symbols become more complex, they take longer to draw, are more prone to error and take longer to recognize. That is, the main benefits of adopting the technique in the first place - fluency and speed - are defeated.

It is often best to adopt a set of predefined symbols, each of which can be specified by a continuous line (thereby providing for maximum lexical compactness). One main area for future research concerns how to obtain optimal performance within these constraints.

TYPING
Typing a command to select and position shapes was one of the techniques implemented. The syntax used was:

```
<command> <X val> <Y val>
```

where the command was one of 's', 'c' or 't' for square, circle, and triangle, respectively. The X and Y values specified the coordinated over which the shape was to be positioned.

[3] This is of special concern in systems designed for casual users.

Typing points out that three tokens are required to fully specify the task. In the previous techniques we have been implicitly treating the position as a single token. The importance of this observation is to point out how effectively the appropriate interaction can bind elements into a single unit. What we described with the character recognizer with respect to selection and positioning, we had been doing all along with the specification of the X and Y values of the position.

While we would sometimes like to believe that graphics solves all problems, the current exercise reminds us that sometimes it is better to type.

If a user was given a picture made up of squares circles and triangles to reproduce using each of the techniques described, it is most likely that typing would lead to the slowest task performance. However, if the task was reformulated and the picture was presented as a list of numeric coordinates (as is often the case in the real world), the typing technique would be the fastest (given a skilled operator). The reason is that the means of performing the task has a good cognitive "fit" with the task formulation. Similar results would result in cases where a high degree of accuracy was required.

The previous points lead us to a more general comment. Each of the techniques described has different strengths and weaknesses, *and there most likely exists a task for which each is optimal and each is abysmal*. Which technique is appropriate is always a function of the task to be performed and how it is formulated.

FUNCTION KEYS

The final technique to be presented involves pointing to the position where the shape is to be located and pressing one of three function keys. In our study, the function keys (one for each of the square, circle and triangle) were mounted on the back of the tablet puck. For free-hand layouts, this was the fastest of all of the techniques tested.

From this part of the study, several important questions remain unanswered. Apart from leaving one hand free for other tasks, is there a speed advantage to using the same hand for pointing and selecting? When and under what constraints? Also, when does the speed advantage of function keys break down? Can this be improved by using chording keys, and if so, at what price (for learning and remembering)?

One approach which we have both seen and tried is to use a puck with only three or four buttons, but change their meaning in different contexts. Our experience is, however, that this is confusing to novice users and leads to a high number of errors. The reason is that "different contexts" means different modes, and as Tesler (1981) has argued, we should be eliminating rather than emphasizing modes. Our conclusion is to fix the function for each key. To get maximum benefit, therefore, we choose the functions to be ones which are both global and frequently used.

One final point, the example provides a good opportunity to point out one difference between a tablet puck *vs* stylus: the puck has function keys and can therefore be used as a selector simultaneously with being used as a pointer. As with the character-recognizer, we see how the pragmatic component can lead to a binding of associated functions so as to support a more articulate and fluent means of task performance.

CONCLUSIONS

A number of different techniques for performing selection-positioning tasks have been presented. Each was shown to have differing properties in terms of costs and benefits. One conclusion has been that which of these techniques is most appropriate in a given context depends on the task to be performed and how that task is formulated.

If it is important to understand the differences among the various techniques, then it is perhaps even more important to acknowledge the value of the technique used to bring these properties to light. We believe that creating a testbed where these techniques can be refined and compared in a common, manageable environment has resulted in an acceleration of our understanding of the various issues and their respective importance. Some of the questions which have arisen will now be pursued in more detail. Finally, we intend to use the same test bed to investigate other tasks, and to develop better tools for prototyping such toy systems and evaluating their performance.

ACKNOWLEDGEMENTS

The work reported in this chapter owes a great deal to the work of Sanand Patel, Howard Ray and Anthony Kroll. Patel implemented the menu system which was used, while Ray and Kroll implemented the techniques described. In addition, I am indebted to Alain Fournier for his helpful comments during the preparation of this manuscript. Finally, I would like to acknowledge the financial support of the Social Sciences and Humanities Research Council of Canada and the National Sciences and Engineering Research Council of Canada.