Touchscreen Field Specification for Public Access Database Queries: Let Your Fingers Do the Walking

Andrew Sears
Yoram Kochavy
Ben Shneiderman

Human-Computer Interaction Laboratory
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract: Database query is becoming a common task in public access systems; touchscreens can provide an appealing interface for such a system. This paper explores three interfaces for constructing queries on alphabetic field values with a touchscreen interface; including a QWERTY keyboard, an Alphabetic keyboard, and a Reduced Input Data Entry (RIDE) interface. The RIDE interface allows field values to be entered with fewer "keystrokes" (touches) than either keyboard while eliminating certain errors. In one test database, the RIDE interface required 69% fewer keystrokes than either keyboard interface.

Introduction

Searching a database is a daily task. Looking up a telephone number in the phone book or looking up an item's description and price in a store's catalog are examples of a database query.

The traditional input device for database queries is a keyboard. Users formulate a queries by selecting a field, a relational operator, and then values for the field, possibly followed by additional specifications (i.e. another field, operator, and value) (Ageloff, 1988; Date, 1988, Gittins, 1986). There are many places where it is easy to make an error in a such a system. Users may type a field name that is not valid (either not in the database or just misspelled), select an operator that is not applicable at that point in the query, or type a value for a field that is misspelled or inappropriate such as a number when a person's name is required (Welty, 1985).

Touchscreen interfaces are becoming more common for public access systems. The reasons for this success include rapid performance, ease of learning, lack of moving parts, and durability (Pickering, 1986; Shneiderman, 1987; Stone, 1987; Muratore, 1987; Priest & Pfauth, 1981).

Research has resulted in solutions for many of the problems that plagued early touchscreens including a lack of precision, high error rates, and arm fatigue. The lack of precision has been addressed by both industry and academic researchers. Elographics has developed touchscreens with resolution as high as 4096*4096 (Elographics, 1983). Further research has increased precision and enabled the accurate selection of single pixel targets (Sears & Shneiderman; 1989). Several researchers have also addressed the problem of high error rates. Some solutions explored the use of alternate selection strategies (Potter, Weldon & Shneiderman, 1988; Potter, Berman & Shneiderman, 1989; Murphy, 1986). A more recent attempt involved smoothing the data received from the touchscreen to compensate for the lack of stability in hardware (Sears & Shneiderman, 1989). Arm fatigue was reduced and difficulties with the reading angle were resolved in a study by Weisner (1988). Weisner allowed users to pivot the monitor to the angle they preferred.

These improvements have made the touchscreen well suited to public access systems and it is becoming more popular with designers of such systems. This paper explores database query using a touchscreen as the sole input device and concentrates on the problem of specifying alphabetic values for fields.

Previous Research

Tennant and Ross (1983) proposed a menu driven natural language interface for database queries. This system presented users with a set of menus that contained only valid entries at each stage of the query. Users selected items with either a mouse or by using the arrow keys and pressing return. This eliminated the selection of invalid fields, operators, and in some cases, inappropriate data values. Any query that could be formed, could be interpreted by the system without errors. However, this system did not address the issue of selecting a field value when there were many possibilities (e.g. entering a person's name where all possibilities cannot be listed conveniently).

Converse et al. (1988) reviewed a keyboard based computerized shopping mall directory. This system allowed shoppers to locate stores by name, product or location. With a single key press using a special keyboard, shoppers could view a numbered list of stores in a particular part of the mall, or stores that sold a particular type of product. Once this list was presented, shoppers could then select a store by pressing the corresponding numbered key on a different section of the keyboard. This system reduced the possibility of selecting an inappropriate entry by a menu-like use of the keyboard. There are several disadvantages associated with keyboard based systems. The first is that there are extra keys on the keyboard that may not valid at a particular stage of the query that may be selected by users producing an undetermined result. Another disadvantage is the additional cognitive load required to translate an item on the screen into a specific key press. Both of these problems can be solved by a touchscreen interface. Touchscreen interfaces allow the elimination of inappropriate keys and reduce the cognitive load required to select an item by allowing the user to simply touch the desired item.

Touchscreen Interfaces

Designing a user interface to take advantage of a touchscreen is quite different than designing traditional keyboard interfaces. Touchscreens allow the definition of selectable regions to act as "keys." Touchscreen interfaces can be customized at each stage of the interaction; when a key becomes unnecessary it can simply be removed from the interface, eliminating the possibility of an inappropriate selection of that key (Priest & Pfauth, 1981). Keys on a keyboard can be inactivated, however users may still press the key expecting some response.

Touchscreens allow a natural method for selecting items from the screen (Shneiderman, 1987). Users simply point to an item on the screen, and it is selected. Unlike touchscreens, keyboards require items on the screen to be mapped into sequences of key-presses. This requires additional cognitive effort from users and may result in higher error rates and slower performance.

The rapid selection and ease of use of touchscreens often result in high user satisfaction. Other benefits include a minimal learning time and the guidance a touchscreen system can provide (Priest & Pfauth, 1981). However, most touchscreens allow only a single touch at a time, making resting your hand or fingers on the screen impossible. This may have significant implications on tasks that involve typing or precise selections.

Touchscreen interfaces can be customized easily for the specific type of interaction. For example, selecting a specific color can be a difficult task. Describing the color you wish to select is not easy using words, however touching the approximate color on a color chart is quite simple. Another example is entering numbers. If the user is entering a telephone number, a touch sensitive telephone keypad could be presented. On the other hand, if users are entering numeric values, a calculator-like keypad could be presented.

Touchscreen Interfaces for Field Specification

In most database queries a field value must be specified. For touchscreen interfaces for database queries three distinct categories have been identified:

(1) N or fewer possible values,
(2) More than N possible values,
(3) Special field,

where N is the number of items that can be displayed on and selected from the screen at one time.

The third category may include fields such as colors, shapes, or numbers. Colors may be selected from a color spectrum presented on the screen by simply touching the desired shade. Shapes could be selected by having users draw the desired shape on the screen with their finger. Entering numbers could be done using either a calculator like keypad or a telephone keypad, depending on which is more appropriate for the task. Special fields may require a unique interface for optimal interaction.

Fields that fit into the first category have a limited number of possible values. All values can be presented on the screen simultaneously in a menu like format, and users simply touch the desired item. For fields in this category, a menu like design is usually appropriate for entering values.

Fields that fit into the second category are the main topic of the remainder of this paper. These fields have N+1 or more possible values. An example of such a field is the author's name on a library's online catalog system. There is no way that all possible names could be listed on the screen at one time in a usable manner. Other solutions are needed to enter values for fields that fit into this category. Several possible solutions will be discussed in this paper, including: a QWERTY keyboard, an Alphabetic keyboard, and a Reduced Input Data Entry (RIDE) interface. For both keyboard interfaces, the keyboard is presented on the touchscreen surface and users "type" letters by touching the appropriate selectable areas. The RIDE interface uses successive approximation to reduce the number of possible inputs at each stage of the interaction. The system uses knowledge of the valid values in the database, and the data entered up to that point in the interaction to eliminate inappropriate options. This allows values to be entered with fewer inputs and errors than is otherwise possible. In some cases, the RIDE interface requires less than 31% of the inputs needed by a keyboard interface (Table 6).

Touchscreen Interfaces for Fields with Many Values (Category 2)

In this section, three solutions to entering values for fields with more than N possibilities (Category 2) will be discussed. To simplify the discussion, a representative task of locating a person's phone number in an electronic phone book was chosen.

Three possible interfaces were identified: a QWERTY keyboard, an Alphabetic keyboard, and a Reduced Input Data Entry (RIDE) interface. Other possibilities include scrolling through a complete list of values, or using an auto-completion strategy that presents additional letters when there is no ambiguity. Scrolling through a list of values may be sufficient when there is a limited number of possibilities. When there are many possible values, such as last names in a telephone book, this solution would require extra work on the part of users. Auto-completion strategies could assist users in reducing the number of possibilities, however, these strategies require complete unambiguity before benefiting users. Strategies could also be devised that used a keyboard to start entering a value and once enough information has been specified the possible values could all be presented simultaneously, as they would for a Category 1 field.

QWERTY Keyboard Interface

The QWERTY keyboard interface resembles the standard keyboard on most computers and typewriters, having the Q,W,E,R,T, and Y keys across the top row of keys, but is presented on the touchscreen. This keyboard was originally designed to place frequently typed letters far apart in order to prevent manual typewriters from jamming. This design has persisted, even with the introduction of more efficient, faster keyboards (Montgomery, 1982; Grudin, 1989). The main advantage of the QWERTY keyboard is that it is well known by computer users and typists.

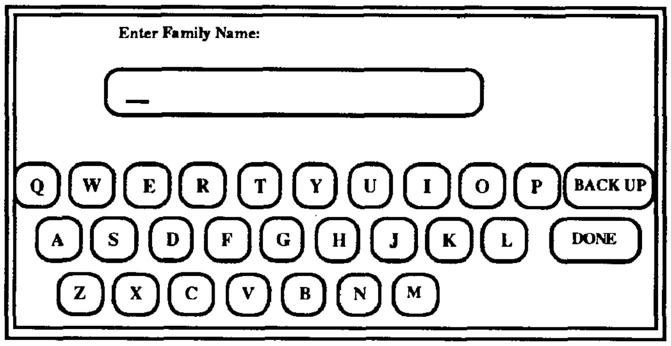

Figure 1 - QWERTY keyboard layout

Using the QWERTY interface (Figure 1), users would simply type (touch) the desired value on the touchscreen, followed by the "done" key to indicate the string is complete. To input a string of length L, L+1 inputs are required; one input per character followed by the "done" key. Inputting a string of length L using a QWERTY keyboard requires the following:

- minimum inputs = L+1+(2*n*e),
- maximum inputs = L+1+(2*n*e),
- average inputs = L+1+(2*n*e),

where e = the number of errors made, and n = the average number of letters typed after an error before users realize that the error was made (including the incorrect letter). For each of these n letters two inputs are necessary to correct the error. The first is the incorrect letter, and the second is the "back up" that is necessary to remove the error.

There are several problems with using a QWERTY keyboard interface. The first is that inexperienced users must search the keyboard for the appropriate key, the location of a key is unpredictable. Novices are forced to search the keyboard to locate each key as they need it (Nicolson & Gardner, 1985). In addition, a QWERTY keyboard interface does not prevent misspellings or inappropriate values from being entered.

Issues that must be addressed when using a QWERTY keyboard layout include color and size coding of keys. Color coding could be used to distinguish vowels (A, E, I, O, and U) or common keys (S, N, T, R, and L). These keys could be in a different color making them easier to locate. Size coding could be used to distinguish commonly used from rarely used keys. Common keys could be larger than average, making them easier to locate and touch, while uncommon keys (Q, X, and Z) could be smaller, making them less likely to be selected by accident.

Alphabetic Keyboard Interface

An Alphabetic keyboard interface has the keys arranged in alphabetical order typically in two to five rows on the touchscreen. The main rationale for alphabetic keyboards is the consistency and predictability they provide. Everyone learns the alphabet in the same order, making an alphabetical layout consistent with prior knowledge, reducing the amount of new information that needs to be learned. Many studies have been performed to compare alphabetic and QWERTY keyboards. Nicolson and Gardner (1985) provide an example where novices perform better with an alphabetic keyboard, while other researchers have indicated that an alphabetic arrangement may never be advantageous (Norman & Fisher, 1982; Potosnak, 1988).

Most studies that have shown superior performance for the QWERTY layout have used tasks that are normally performed by touch-typing on a keyboard. Long typing tasks, such as entering a sentence, paragraph, or entire paper, may be slow on a touchscreen for several reasons. Fast typing typically requires that typists keep their fingers on or near a "home row" of keys without having to watch them. This is difficult on a touchscreen for two reasons; resting your fingers on the screen is not possible with most touchscreens, and there are no key edges on a touchscreen, making it difficult to know when your hand has slipped from the "home row." Since we are dealing with a touchscreen and tasks that involve entering relatively short strings, most studies do not apply to these tasks.

The second interface proposed used an alphabetic layout for entering alphabetic characters (Figure 2). Using this interface, users would simply type (touch) the desired value on the touchscreen, followed by the "done" key to indicate the string is complete. To input a string of length L, L+1 inputs are required; one input per character followed by the "done" key. Inputting a string of length L using an alphabetic keyboard requires the following:

- minimum inputs = L+1+(2*n*e),
- maximum inputs = L+1+(2*n*e),
- average inputs = L+1+(2*n*e),

where e = the number of errors made, and n = the average number of letters typed after an error before users realize that the error was made (including the incorrect letter). For each of these n letters two inputs are necessary to correct the error. The first is the incorrect letter, and the second is the "back up" that is necessary to remove the error.

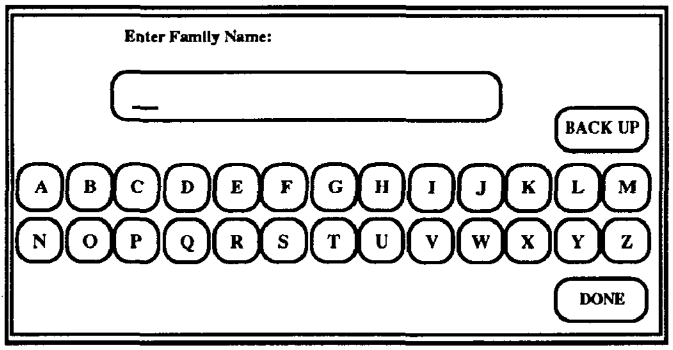

Figure 2 - Alphabetic keyboard layout

There are several problems that occur when using an alphabetic keyboard interface. The first is that users who are familiar with typewriters or computer terminals must adjust to the new layout. However, the layout of the keyboard is consistent with prior knowledge of the alphabet, making it possible to predict where a key is located. One additional disadvantage of an alphabetic keyboard interface is that it does not prevent misspellings or inappropriate values from being entered.

Other issues that must be addressed when using an alphabetic keyboard layout include the number of rows and columns of keys, color and size coding of keys. Color and size coding effects have been discussed in the section about the QWERTY keyboard interface.

There are many possibilities for the number of rows and columns of keys. There could be two rows of thirteen keys, three rows with eight or nine keys, or five rows with five or six keys. Several factors may determine which layout is superior. Items at the end of each row of keys may act as "anchors" when searching for a particular key. These items would be located faster, and if there are fewer keys between anchors (as there are in the 5x5+1 layout) these items should be distinguished faster. On the other hand, having many rows of keys may disturb users who are familiar with the standard QWERTY keyboard. The similarity of the three row layout to the standard QWERTY keyboard may result in a negative transfer of knowledge of key location resulting in increased scan time and error rates. Of course when designing an alphabetic keyboard layout the position of the backspace and done or return keys must also be considered.

Reduced Input Data Entry (RIDE) Interface

The typical way of entering values when there are many possibilities (Category 2) is to type them. A RIDE interface reduces entering values from a typing task to a menu selection task.

The RIDE interface initially presents users with a list of all possible one letter prefixes (Figure 3a). Users select the first letter for the desired input, and the system then presents a list of all possible two letter prefixes (Figure 3b). Users then select the two letter prefix for the desired input value, and so on. If at any stage in the process RIDE can expand more than one character, while producing N or fewer possibilities, it will do so (Figure 3c). This process continues until the number of possible values is N or less (N = number of items that can be displayed on and selected from the screen at one time) (Figure 3c). Although this could be done with either keyboard interface, users typically type the entire value once they start on a keyboard. Having the interface change in the middle of entering a value may make users uncomfortable. With the RIDE interface, it would be expected that at some point a list of valid values would be presented. This may occur as early as the first selection if there are N or fewer values (Category 1). If at any point, the leading string for the desired value is not presented as an option, users can determine that the desired value is not contained in the database. Users can back up at any time in the process. Instead of backing up a single character, RIDE backs up to the previous stage in the process. This may only be a single character, or the entire string.

The third interface, RIDE, offers many advantages. RIDE leads users along a path to entering only valid values. Misspelled values are limited to alternate spellings that appear in the database, and inappropriate values, such as numbers for an alphabetic field, cannot be entered. Inputting a string of length L using a RIDE interface requires the following:

- minimum inputs = 1+(2*n*e),
- maximum inputs = L+1+(2*n*e),
- average inputs = varies from database to database,

where e = the number of errors made and n = the average number of inputs after an error before users realize that the error was made (including the initial incorrect input). For each of these n inputs two inputs are necessary to correct the error. The first is the incorrect input, and the second is the "back up" that is necessary to remove the error.

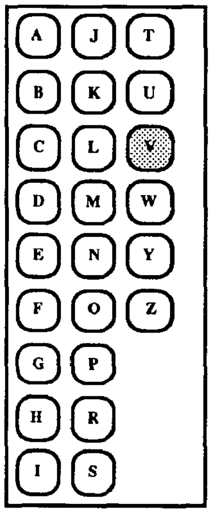

Figure 3a - One Letter Prefixes

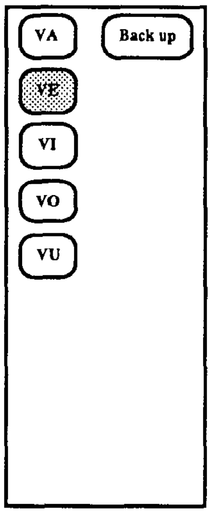

Figure 3b - Two Letter Prefixes

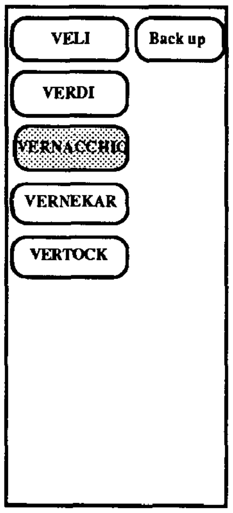

Figure 3c - After Two Selections Entire Names Can be Listed

There are many factors that influence the average number of inputs necessary to select an item using the RIDE interface. These include the length of the desired string, the size of the database (how many possible values), and the uniqueness of the leading substrings of the data. The influence of these factors will be discussed in the following section. The average number of letters typed after an error before the user realizes that the error was made should be less for the RIDE interface than it would be for either keyboard. This is due to the fact that users are only presented with valid options at each stage when using the RIDE interface, making it easier for users to realize that an error has been made.

There are several problems with using the RIDE interface. First, users who are familiar with typewriters or computer terminals must adjust to the new interface. Although the interface is consistent in placing all valid items on the screen in alphabetical order, users can not predict the location of the desired item and have to visually scan each screen.

The time to scan the screen slows performance when using the RIDE interface. Variables that influence this scan time include size of the keys and the arrangement of the options on the screen. The size of the keys must be appropriate, allowing users to easily touch the key, while not being so large that they increase scan time. There are many factors that determine the arrangement of the options that may also influence the visual scan time. These factors include the number of rows and columns of keys (3x9, 5x5, 2x13, etc.), the order that options are placed on the screen (top to bottom or left to right), whether the arrangement of the keys remains constant with unavailable keys removed or all available keys are displayed with no gaps. Displaying the available keys without gaps reduces the area that the user must scan, however, it also decreases the consistency of the layout.

When the database being accessed is static, or changes infrequently, indices can be calculated into the database while users are not using it, resulting in fast search times. This will make the time the system needs to search the database for a prefix insignificant compared to the time users spend thinking or entering information.

Discussion

Each of these interfaces has been implemented and informally evaluated with a database of approximately 2500 names. The maximum number of names displayed on the screen at one time, N, was set to nine, allowing all names to be presented in a single column on the screen. Due to the significant time necessary to redraw the screen for the RIDE interface, the QWERTY keyboard was slightly faster than RIDE, while the Alphabetic keyboard was the slowest. Preliminary calculations indicate that if the time necessary to redraw the screen were reduced, the RIDE interface would require slightly less time than the QWERTY keyboard. In addition, the RIDE interface appears to result in significantly fewer errors than either of the other interfaces, as was predicted. Preprocessing of the data allowed rapid searches on prefixes, making the time the computer spent searching the database insignificant.

Each of the three interfaces we discussed has advantages and disadvantages.While both keyboard interfaces provide feedback indicating what users have just selected, the RIDE interface provides more direct feedback. The feedback from a keyboard interface is located at a point different than where the selection is made. RIDE provides feedback where the next selection will be. At each point in the interaction all possible inputs indicate exactly what has been entered up to this point.

The power of the RIDE interface is best demonstrated when entering multiple fields for a particular entry in the database (e.g. last name then first name). After the last name has been specified the number of possible first names has been significantly reduced. This will typically result in first names (or other additional fields) being entered with a minimum of inputs.

There are many tasks for which the RIDE interface may allow users to perform faster and with fewer errors than either keyboard. These include tasks where there are a limited number of valid values, values have unique leading substrings, and value lengths are long and unique.

Using the RIDE interface for a limited number of values significantly reduces the number of inputs necessary to select a value, and eliminates the possibility of entering an invalid values. When there are a limited number of values, the RIDE interface narrows the possibilities faster, allowing values to be selected with fewer inputs (Table 6).

Another factor that can drastically improve the performance of RIDE is the uniqueness of the possible values. If the last K characters of a string of length L are identical, the maximum number of inputs necessary to select a value becomes L-K+1. If on the other hand, the first J characters of all of the strings are identical, the first strings that are presented to the user will be of length J+1, reducing the maximum number of inputs to L-J. This factor is a major consideration when evaluating the efficiency of the RIDE interface for a specific task.

A third factor that effects performance of the RIDE interface is the length of the strings being selected. If the strings are relatively long, the RIDE interface significantly reduces the number of inputs necessary. There is a strong interaction between these three factors.

To compare the number of inputs necessary using each interface, statistics for three databases containing the last names of students at the University of Maryland were calculated. The first database contained approximately 16000 names (Tables 1 and 2), the second contained approximately 1600 names (Tables 1 and 3), and the third contained 160 names (Tables 1 and 4). The maximum number of names that could be displayed on a single screen, N, was set to nine allowing all names to be presented in a single column. These databases provide some insight into how the RIDE interface will perform compared to a keyboard for various size databases. This data assumes that there were no errors made when entering the names. It is anticipated that if errors were made the results would prove even more favorable for the RIDE interface.

The data in Tables 2, 3, and 4 clearly indicates that the RIDE interface requires significantly fewer inputs to enter a name on average. The average number of inputs necessary for the RIDE interface is only 58% of the number required for the keyboard interfaces for the database of 16000 names, 47% for the database of 1600 names, and 31% for the database of 160 names. The minimum and maximum number of inputs necessary for the RIDE interface are also significantly less than for the keyboard interfaces. For the database of 160 names the maximum number of inputs users would ever have to make is 4 with the RIDE interface compared to 14 with either keyboard. It is clear that as the database size decreases the benefit from the RIDE interface increases. Entering additional information about a student, such as the first name, would result in an even greater benefit when using the RIDE interface. In most instances the first name could be specified in one or two inputs.

Performance statistics can be calculated using these formulas when deciding which interface to use for accessing a particular database. These statistics guide designers in predicting user performance with each interface.

	Size of Database		
	15818	1583	160
Name Length (L)	Number of Names of Length (L)		
3 or less	0	0	0
4	1197	125	12
5	2621	260	29
6	3556	357	33
7	3416	355	42
8	2333	226	28
9	1379	118	8
10	788	80	5
11	306	33	1
12	116	14	1
13	49	5	1
14	26	5	0
15	20	3	0
16	4	0	0
17	4	0	0
18	3	1	0
19 or more	0	0	0

Table 1 - Number of Names by Length

Database 1 - 15818 names

Inputs per name	RIDE	QWERTY	Alphabetic
Average	4.55	7.85	7.85
Minimum	2	5	5
Maximum	7	19	19

Table 2 - Inputs per Name by Interface

Database 2 - 1583 names

Inputs per name	RIDE	QWERTY	Alphabetic
Average	3.70	7.84	7.84
Minimum	2	5	5
Maximum	5	19	19

Table 3 - Inputs per Name by Interface

Database 3 - 160 names

Inputs per name	RIDE	QWERTY	Alphabetic
Average	2.35	7.67	7.67
Minimum	2	5	5
Maximum	4	14	14

Table 4 - Inputs per Name by Interface

Conclusion

This paper explored three possible interfaces for public access database queries in a touchscreen environment. Touchscreen versions of keyboard interfaces have some advantages, and the QWERTY design seems superior to the Alphabetic for most tasks and users. Refinements are possible for all three versions, but the Reduced Input Data Entry interface reduces the number of inputs, at the expense of increased perceptual and cognitive load. The advantage of the RIDE interface increases as the database size decreases.

Acknowledgements

We would like to thank Christine Van Eseltine and Richard Doyle Jr. for performing the initial experiment providing insight into the performance of the three interfaces we discussed. We would also like to thank NCR for partial support of this work.

References

Ageloff, R. (1988), *A Primer on SQL*, Timer Mirror/Mosby College Publishing, St. Louis.

Converse, S., Fedorko, M., Hammontree, M., Montero, C., Thornton, C., & Zwaga, H. (1988), Where can I Find ... ? An Evaluation of a Computerized Directory Information System, Unpublished manuscript.

Date, C.J. (1988), *A Guide to The SQL Standard*, Addison-Wesley, Reading, MA.

Elographics, Inc. (1983), Model E271-60, Oak Ridge, TN.

Gittins, D. (1986), *Query Language Systems*, Edward Arnold, Baltimore, MD.

Grudin, J. (1989), The Case Against User Interface Consistency, *Communications of the ACM V32*, October, 10, 1164-1173.

Montgomery, E. (1982), Bringing Manual Input into the 20th Century: New Keyboard Concepts, *IEEE Computer V15*, March, 3, 11-18.

Muratore, D.A. (1987), Human Performance Aspects of Cursor Control Devices, MITRE Corporation Working Paper 6321, Houston, TX.

Nicolson, R. and Gardner, P. (1985), The QWERTY keyboard hampers schoolchildren., *British Journal of Psychology V76*, 525-531.

Norman,D. and Fisher, D. (1982), Why alphabetic keyboards are not easy to use: Keyboard layout doesn't much matter., *Human Factors V24*, 509-515.

Pickering, J. (1986), Touch-sensitive screens: The Technologies and their applications, *International Journal of Man-Machine Studies V25*, 249-269.

Potter, R., Weldon, L., and Shneiderman, B., (1988). Improving the accuracy of touch screens: An experimental evaluation of three strategies, *Proceedings of the Conference on Human Factors in Computing Systems*, ACM SIGCHI, New York, 27-32.

Potter, R., Berman, M., and Shneiderman, B., (1989). An experimental evaluation of three touch screen strategies within a Hyperties database, *International Journal of Human-Computer Interaction V1*,1, 41-52.

Potosnak, K., (1988). Keys and Keyboards, *In Handbook of Human-Computer Interaction* (Helander ed.), Elsevier Science Publishers, North-Holland, 475-494.

Priest, J. and Pfauth, M. (1981), Touch Screen Devices: Industrial Engineering Considerations of an Emerging Technology.

Sears, A. and Shneiderman, B. (1989), High Precision Target Selection: A Comparison of Touchscreens and a Mouse, University of Maryland Computer Science Technical Report Number CS-TR-2268.

Shneiderman, B. (1987), *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, MA.

Stone, M.D. (1987), Touch-Screens for Intuitive Input, *PC Magazine*, August, 183-192.

Tennant, R. and Ross, K. (1983), Usable Natural Language Interfaces Through Menu-Based Natural Language Understanding, ACM CHI'83 Proceedings, 154-160.

Weisner,S. (1988), A Touch-Only User Interface for a Medical Monitor, *Proceedings of the Human Factors Society*, 435-439.

Welty, C. (1985), Correcting user errors in SQL, *International Journal of Man-Machine Studies V22*, 463-477.