CHAPTER 9
Symbolic Input

We turn our attention in this chapter to the task of symbolic input, a 3D interaction task that has been studied relatively little compared to travel, wayfinding, selection, manipulation, and system control. Symbolic input is the task in which users communicate symbolic information (text, numbers, and other symbols or marks) to the system. Everyone who uses computers performs symbolic input tasks constantly—writing email messages, entering numbers into a spreadsheet, composing documents, and so on. The importance of symbolic input in 2D interfaces is clear, but why is symbolic input an important task in 3D UIs?

9.1. Introduction

Symbolic communication—the use of abstract symbols to represent objects, ideas, concepts, quantities, and the like—is one of the crowning achievements of human civilization. It allows us to provide and obtain information precisely and concisely; it allows information to be persistent; it provides for methods of thought not possible without symbols. Imagine what it would be like to live in a world without language or mathematics! Symbolic communication pervades everything we do.

It is strange, then, that symbolic communication in most 3D interfaces is either nonexistent or limited to one-way communication. Many 3D applications, such as architectural walkthroughs, neither present nor

accept symbolic information; rather, they present a purely geometric and visual world for the user to perceive. Of the 3D applications that do include symbolic information, most provide symbolic output only. That is, text, numbers, or speech are embedded in the environment. This might take the form of labels on virtual buttons, a legend on a map, a numeric display of the user's coordinates, or audio help, to name a few examples. Symbolic input, however, is rarely present.

9.1.1. Why is Symbolic Input Important?

Nevertheless, we claim that this lack of symbolic input in 3D interfaces does not mean that the task is unimportant for 3D applications, but rather that usable and efficient techniques for this task are difficult to design and implement, which has caused developers to largely avoid the issue. We posed the question, Is text/number entry an important task for VEs? on the 3D UI mailing list, an online community whose membership includes most of the key researchers in the field of 3D interaction worldwide. This question sparked the longest discussion thread ever on the mailing list. While no one envisioned word processing systems in a VE, all of the respondents agreed that text input for immersive VEs was an important research area. Many of them also suggested novel techniques that might be attempted, including split, wearable keyboards and hand gestures.

9.1.2. Scenarios of Use

In order to further demonstrate the potential importance of symbolic input for 3D UIs, we discuss several realistic scenarios of use. The scenarios involve immersive VEs or augmented environments where a general text/number-input technique would be needed and where speech input alone would likely not suffice. The list of scenarios includes the following:

- *Design annotation:* Suppose an architect develops a new design for the atrium of a building. He sends the 3D model to his clients, who walk through the new design in a VE. During the walk-through, clients wish to annotate the design with questions, suggestions, or change orders. Speech could be used for this application, but a text/number entry technique would allow for greater precision, the ability to edit the annotation, the ability to use nonspeech elements such as URLs or numeric measurements, and better understanding of the annotation when it is read.

- *Filename entry:* Many immersive systems that allow the user to add new elements to the environment or modify objects in the environment need the ability to open and save files, similar to text editors or other 2D applications. Since filenames are often nonwords, a general symbolic input technique is needed for this task.
- *Labeling:* Another type of annotation is the task of adding labels to objects in virtual or augmented environments. For example, the user of a mobile augmented reality system might be able to provide further augmentation by adding labels to objects she sees in the world.
- *Precise object manipulation:* In certain situations, the typical freeform object manipulation techniques (Chapter 5) may not provide enough precision. For example, if the user is designing an engineering model that will be used to produce a physical part, precise position, thickness, length, and other dimensions will be needed. Rough manipulation could be performed with a direct technique, but the final values may need to be entered numerically. This scenario is analogous to current desktop 3D modeling and CAD software, which typically includes both direct manipulation and numeric field entry for specifying object properties.
- *Parameter setting:* Many applications require a numeric entry technique to set some parameter needed by the system. For example, in a visualization of airflow around an airplane fuselage, the user might need to specify the aircraft velocity or the angle of the flaps. An immersive game might allow the user to set the maximum number of virtual opponents. Specifying RGB color values might be necessary to match two colors in a 3D design environment.
- *Communication between users:* When multiple users share a 3D environment, they usually require some method of communication. In collaborative VEs (CVEs), speech is most often used for this task, but some CVEs would benefit from a nonspeech text/number-entry technique. For example, if two engineers are collaborating to assess the structural integrity of a proposed bridge design, they could use a shared "whiteboard" to work out equations or note possible design alternatives. Unlike speech, this communication would be precise and persistent. The popularity

of text-based chat systems also illustrates that people accept textual as well as spoken communication.

- *Markup:* Many of the scenarios above would provide further benefit if users could augment their textual input with markup features, such as underlining, italics, and highlighting. For example, the architectural annotations described earlier might carry more force if the client could highlight or use bold text to emphasize important points.

9.1.3. Brief History of Symbolic Input

The canonical symbolic input device for computing systems is, of course, the keyboard. In particular, the so-called QWERTY layout (named after the first six keys on the second row of the keyboard) is the accepted keyboard for almost all typewriters, computers, and other text-entry systems. The history of the QWERTY keyboard is complex and interesting, and a complete treatment of it is beyond the scope of this book, but we summarize it below (see the following references for more thorough surveys: Cooper 1983; Dvorak et al. 1936; Yamada 1980).

The QWERTY layout was first devised by Christopher Sholes in 1873. Sholes had been working to find a typewriter keyboard layout that would minimize mechanical jams caused by the arms that strike the paper to print a character. Because of this, it is often assumed today that Sholes designed the QWERTY layout intentionally to slow typists down, but this is not the case. Rather, the layout places letters that commonly appear consecutively on opposite sides of the keyboard. Since the corresponding arms would also be on opposite sides, there would be fewer jams, but since typists could also use fingers on different hands to type the two consecutive letters, there would be an increase in typing speed as well. Nevertheless, the layout is based on only rough statistics about the frequency of letters and their tendency to appear consecutively.

Since the development of the QWERTY layout, numerous layouts have been devised that are statistically much more optimal in terms of letter frequency and the use of the two hands in rapid sequence. The most famous of these is the Dvorak layout. It has even been claimed that a random ordering of the letters on the keyboard would almost always be better than QWERTY. Despite intense effort in this area, however, only minimal gains in efficiency have been found, and then only when typists practice the layouts over a long period of time. Thus, the QWERTY

layout became a standard that was so entrenched that it has never been displaced.

Of course, the QWERTY layout applies only to keyboards meant to produce text using a Latin alphabet (for languages such as English). There are other keyboard layouts for other alphabets, such as Cyrillic, Japanese kanji, and Mandarin. However, because of the limited scope of this book, we limit our discussion of text input for 3D interfaces to the Latin alphabet.

Since computing devices have become smaller, more portable, and more ubiquitous, research in keyboard design in particular and text-input techniques in general has once again become prominent. We now have keyboards used for text input on personal organizers (PDAs), cell phones, automatic teller machines, and palmtop computers. Many of these still default to the QWERTY layout, since everyone is familiar with it. Most were not designed for 10-finger touch typing, but rather assume a single finger or pen will be used. However, some of these devices use alternate keyboard layouts or form factors, and some use a virtual ("soft") keyboard or no keyboard at all. We draw heavily from these domains in our discussion of possible symbolic input techniques for 3D UIs.

9.1.4. Distinctive Features of Symbolic Input in 3D UIs

Symbolic input techniques for 3D UIs will be necessarily different than traditional techniques (keyboards) because of the inherent differences between 3D (non-desktop) and 2D UIs. Put simply, traditional keyboards don't work in 3D UIs because

- Users are often standing.
- Users may physically move about.
- There is usually no surface on which to place a keyboard.
- It may be difficult or impossible to see a keyboard in low-light environments (e.g., in a CAVE) or when the user's vision is occluded (e.g., in an HMD).

These constraints don't apply to all 3D UIs, and there are potential workarounds (e.g., a keyboard can be strapped to the user's waist in such a way that it can be used while the user is standing and walking around), but in general, we need to consider different methods of symbolic input in VEs and augmented environments.

Given these constraints, if symbolic input tasks in 3D UIs were exactly the same as those in 2D UIs, then the 3D symbolic input problem would be a very difficult one to solve. Looking at the list of scenarios in section 9.1.2, however, we see that symbolic input in 3D UIs may be much less frequent than symbolic input in other interfaces. Furthermore, each instance of symbolic input in a 3D UI may be relatively short in terms of the number of symbols entered—as we have said, we do not foresee document or email composition in an immersive VE.

Another aid to the designer is the wealth of information that already exists on symbolic input in nontraditional computing environments. Designers of wearable computers, palmtop computers, PDAs, and even cell phones have had to tackle these issues. Although the usability of some of the symbolic input techniques for these devices is questionable, these domains are an important source of ideas for 3D UI symbolic input.

9.1.5. Chapter Roadmap

The remainder of the chapter is organized as follows. In section 9.2, we look closely at the general categories of symbolic input tasks for 3D UIs. The heart of the chapter, in section 9.3, is a presentation of techniques, both implemented and proposed, for 3D UI symbolic input. A list of design guidelines is presented in section 9.4.

The chapter focuses solely on symbolic input and ignores the issue of symbolic *output* in 3D UIs. We chose not to include output for two reasons. First, our aim in this book is to present methods of 3D interaction, and symbolic output is not strictly an interaction task, since it involves no action on the part of the user. Second, the design of symbolic output in 3D UIs is simply a special application of well-known principles of information presentation and layout. For further information, refer to the literature in this area (e.g., Tufte 1990; Zwaga et al. 1999).

This chapter is also different from the other chapters in Part III in two ways. First, we focus almost exclusively on non-desktop interfaces (VEs and AR), since symbolic input on the desktop is standardized and well studied. Second, there is less information in this chapter on empirical evaluation and performance of the techniques, since there have been very few formal experiments on 3D UI symbolic input techniques.

9.2. Symbolic Input Tasks

As we have stated, the task of symbolic input in general involves the entry of letters, numbers, and other marks and symbols. However, tasks related to the actual entry of the symbols should also be considered, including editing and markup. We discuss the following tasks in this section:

- alphanumeric input
- editing alphanumeric symbols
- markup input

9.2.1. Alphanumeric Input

We use the term *alphanumeric* to denote all of the possible types of symbols users can enter. These include, but are not limited to, alphabetic characters (e.g., a, R), numeric characters (e.g., 2, 6), punctuation (e.g., comma, quotation marks), white space (e.g., space, tab, carriage return), accent marks (e.g., é, ö), and other symbols (e.g., *, %, >, @, #). In 3D UIs, alphabetic and numeric characters, white space, and perhaps a limited subset of punctuation marks, will most often be sufficient. Certain specific applications may require the entry of other symbols as well (e.g., symbols used in mathematical notation). In addition, there are many other alphabets (e.g., Cyrillic, Japanese kanji) that need to be considered for international applications.

9.2.2. Editing Alphanumeric Symbols

Users may make errors in the entry of symbols; they may decide that another word would be more appropriate; or they may want to suggest a change to some symbols input previously. All of these scenarios require the ability to edit existing symbols. Editing subtasks include specifying an insertion point, deleting the previous character, selecting a character or string of characters, changing a character or string to some other character or string, and so on. At a minimum, users must be able to delete the last symbol entered. This alone is sufficient to perform all the editing subtasks mentioned above, but of course this is hardly desirable if you have just finished a paragraph and realize that the first word is misspelled.

9.2.3. Markup Input

Finally, users may wish to enhance their symbolic input by applying additional formatting, styles, or emphasis, which we call *markup* input. Examples of markup input include underlining, italics, bold characters, highlighting, font specification, color specification, line-spacing specification, editing marks, and the like. In typical 2D interfaces, markup is specified either through the use of special tags embedded in the text (as in HTML) or via a GUI (as in most word processors). Markup input is certainly the least studied, and probably the least important, symbolic input task for 3D UIs.

9.3. Symbolic Input Techniques

In this section, we present a wide range of possible techniques for symbolic input in 3D UIs. We classify the techniques as keyboard-based, gesture-based, pen-based, or speech-based.

9.3.1. Keyboard-Based Techniques

The techniques that fall into the keyboard-based category either use a physical keyboard or are based on a keyboard metaphor. Although it may be possible to use a standard, full-sized keyboard in some 3D UIs, we do not explicitly consider them here. The keyboard-based techniques we will cover are

- miniature keyboards
- low key-count keyboards
- chord keyboards
- Pinch Keyboard
- soft keyboards

Miniature Keyboards

Perhaps the easiest way to bring a keyboard into a 3D UI is to miniaturize it so it can be carried or worn. This technique also has the advantage that it can retain the familiar QWERTY layout so that users do not have to relearn the keys' positions. For this reason, miniature keyboards are popular on many consumer-level mobile devices such as PDAs. However,

miniature keyboards are not generally large enough to allow touch typing (or ten-finger typing), so users must type with one or two fingers and cannot use the muscle memory built up from years of keyboard usage.

In a 3D UI, a miniature keyboard may be held in one hand while the other hand types (similar to a palmtop computer), or the keyboard may be worn by the user or attached to the user. An experiment comparing several text input devices for wearable computers found that a miniature keyboard strapped to the forearm of the nondominant hand produced the best performance and user satisfaction (Thomas et al. 1998).

Low Key-Count Keyboards

A second way to make the keyboard small enough to be held in one hand is to reduce the number of physical keys. For example, the keypads on mobile phones can easily be used with one hand by pressing the 12 to 15 keys with the thumb. Because today's mobile phones contain so many features requiring text input (e.g., entering names into a list of contacts), many different symbolic input mechanisms have been developed for them, some of which could be applied in 3D UIs.

The most common phone-based text-input technique uses the standard layout of letters on the numeric phone keypad (a, b, and c are mapped to the 2 key; d, e, and f to the 3 key; and so on). Users press the key with the desired letter and disambiguate the letter by pressing the key multiple times. In other words, pressing 2 once produces the letter a, while pressing 2 twice produces the letter b. This technique is precise and easy to learn, but can be quite inefficient. Speed with this technique can be especially slow when two letters mapped to the same key need to be entered in sequence. For example, pressing the 2 key three times could be interpreted as aaa, ab, ba, or c. In order to make the meaning clear, the system forces the user to wait for a certain amount of time before another letter can be entered using the same key.

Another phone-based technique, typified by the T9 text-input system, only requires that each key be hit once. In order to determine which letter is meant by each key, the system uses a dictionary and tries to match the keystrokes to known words. When the system has a guess for the current word, the word is displayed and the user can either accept the word or continue typing. This technique can be quite efficient if the guessing works well, but typing names, rare words, or nonwords can be difficult. The method of word completion, however, can be applied to any text input technique.

Chord Keyboards

A *chord keyboard* is a physical input device that aims to provide all the functionality of a full-sized keyboard, but using many fewer keys. In order to provide more symbols than there are keys on the device, the user presses multiple keys simultaneously to produce some symbols. The set of keys that are pressed together is called a chord, a name taken from the method of producing a musical chord on the piano by striking multiple keys at once. Figure 9.1 shows a commercially available chord keyboard (the Twiddler2) that has 12 keys and requires no more than two keys to be pressed at once to produce any letter, number, or standard punctuation symbol.

Chord keyboards have been studied extensively in other contexts (Noyes 1983), particularly mobile and wearable computing, and many different layouts and form factors have been tried. In general, users need a great deal of training to become proficient with chord keyboards, since the layout is not related to the standard QWERTY keyboard, but the advantage of one-handed symbolic input may make this training worthwhile in some situations.

In an experiment comparing several symbolic input techniques (chord keyboard, speech, Pinch Keyboard, and pen-and-tablet keyboard) in an immersive VE, a chord keyboard had the slowest performance, the largest number of errors, and the lowest user preference ratings (Bowman, Rhoton

Figure 9.1 *A 12-key chord keyboard. (Photograph courtesy of Doug Bowman)*

et al. 2002). However, this study did not allow a large amount of training time, and the chord keyboard's performance may have improved over a longer set of trials.

Pinch Keyboard

Bowman and colleagues developed a technique for text input in VEs called the Pinch Keyboard (Bowman, Wingrave et al. 2001). It uses Fakespace Pinch Gloves, lightweight gloves with conductive cloth on each fingertip that sense when two or more fingers are touching. The gloves are comfortable to wear, and because of their on/off nature, there is no ambiguity to user actions. This technique also uses a standard QWERTY keyboard layout so that users can take advantage of the typing skill they already have.

The basic concept of the Pinch Keyboard is that a simple pinch between a thumb and finger on the same hand represents a key press by that finger. Thus, on the home row of the keyboard, left pinky represents a, left ring represents s, and so on. In order to use the "inner" keys such as g and h, and to change rows of the keyboard, 6-DOF trackers are mounted on the gloves. Inner keys are selected by rotating the hand inward (Figure 9.2). The user changes the active row by moving the hands closer to the body (bottom row) or farther away (top row). Users calibrate the location of the rows before using the system by indicating the middle of the top and bottom rows while holding the hands palm down. Still, because the trackers have limited accuracy, fairly large-scale motions are required to change rows or select the inner keys, reducing efficiency.

Graphical feedback is extremely important to make up for the lack of visual and haptic feedback provided by a physical keyboard. Two feedback objects are attached to the view. These show the location of each character and the currently active characters (based on hand position and orientation) via highlighting (Figure 9.2). The text entered by the user is also attached to the user's view. In addition, the technique provides audio feedback that lets the user know when a key has been pressed or when calibration is complete.

Finally, the technique includes special gestures for space (thumb to thumb), backspace (ring to ring), delete all (pinky to pinky), and enter (index to index). This set of arbitrary gestures is small enough to be memorized by the user.

In our experience with the Pinch Keyboard technique, novice users type very slowly and deliberately, even though many of them are speedy touch typists. This might suggest that typing skill does not transfer to our

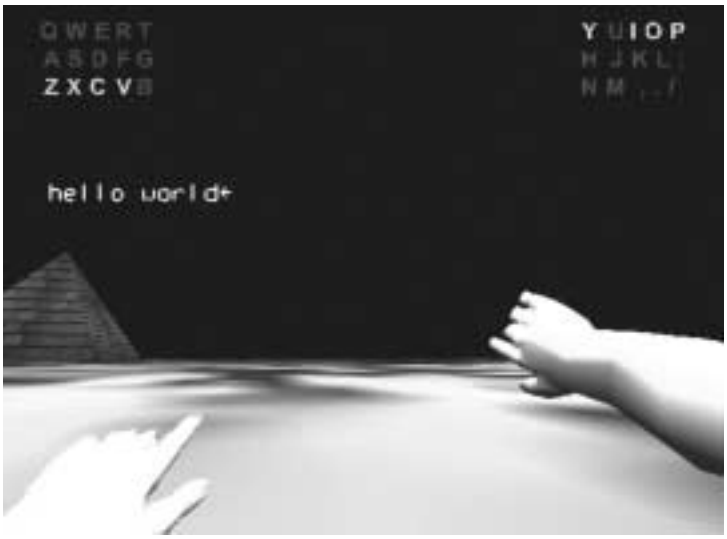

Figure 9.2 *User's view of the Pinch Keyboard. (Bowman, Wingrave et al. 2001, © 2001 by Springer-Verlag; reprinted with permission)*

technique and that training is required. However, all users immediately understand the technique. Expertise is gained not through cognitive training, but rather through motor training, since the movements are different than a standard keyboard. Therefore, novice users can immediately begin typing, although their performance might be poor. Early experimental results (Bowman, Rhoton et al. 2002) indicate that the Pinch Keyboard is slower than speech or soft keyboard techniques, but also that users find the technique easy to learn and satisfying and comfortable to use.

Soft Keyboards

A "soft" keyboard is a virtual device implemented entirely in software. In other words, users press virtual keys instead of physical ones to enter symbols. Some PDAs that lack a physical keyboard include a soft keyboard that is displayed on the screen. Users tap the virtual keys with a stylus or finger to simulate pressing the key. Soft keyboards have the advantages that they can easily be reconfigured for different layouts or alphabets and that they don't require a specialized input device. However, the major disadvantages of soft keyboards are the limitation of single-point input (one finger or pen at a time) and their lack of active and passive haptic feedback. Even if the virtual keyboard lies on a physical

surface, the user cannot feel the contours of the keys to help her find a key without looking or the "click" of the key being depressed to help her know a symbol was entered.

Many types of soft keyboards have been designed that could be used in 3D UIs. For example, the pen-and-tablet metaphor that we have previously seen used for map-based travel (Chapter 6), object manipulation (Chapter 5), and menus (Chapter 8) can also provide a soft keyboard. In Bowman, Rhoton, and Pinho's experiment (2002), this technique was second only to speech in speed and had the fewest errors. Soft keyboards on other devices, such as PDAs, could also be used in 3D UIs, provided that the user can see the device (this technique would not work in HMDs, for example).

A novel virtual keyboard device called the Senseboard (Figure 9.3) may hold some promise for 3D UIs. The concept is that a full-sized QWERTY keyboard can exist virtually on any surface, and users can type with all 10 fingers on this virtual keyboard. This is accomplished by the use of muscle sensors to determine finger movement plus pattern recognition to determine the probable word in ambiguous situations.

Note that virtual keyboards need not be limited to the standard QWERTY layout. In fact, for tapping of a soft keyboard by a pen or single finger, the QWERTY layout is certainly suboptimal (although it will allow better performance by novices). For example, Zhai and colleagues (2000) developed the Metropolis keyboard, a layout for soft keyboards based on quantitative analysis of letter frequency, distance between keys, and so on.

Figure 9.3 *Senseboard virtual keyboard prototype. (Photograph courtesy of Senseboard Technologies AB)*

9.3.2. Pen-Based Techniques

Mobile computing platforms, such as PDAs, often use neither a physical keyboard nor a soft keyboard for symbolic input. Rather, they use pen-based input in which the user writes characters, symbols, or other gestures with a pen/stylus on the device. For example, the Apple Newton allowed users to write in natural cursive handwriting and attempted to recognize entire words. More commonly, single characters are recognized for increased accuracy using either natural characters or modified single-stroke characters, as in the Graffiti alphabet used in the PalmOS. Some such techniques have been applied to 3D UIs, and others may also be appropriate.

We divide pen-based techniques into two categories: *pen-stroke gesture recognition* and *unrecognized pen input*, also called *digital ink.*

Pen-Stroke Gesture Recognition

Recognition of written input is in many ways similar to gesture recognition (section 9.3.3). In pen-based input, the basic unit of recognition is the *stroke*—a pen movement that starts when the pen touches the input surface and ends when the pen is lifted. A complete review of stroke recognition is beyond the scope of this book, but suffice it to say that the same recognition algorithms used in PDAs and other pen-based devices apply equally well to pen-based input in 3D UIs.

A huge number of pen-based symbolic input schemes have been developed. Graffiti, mentioned earlier, is a *character-level* technique, using one stroke per character. Allegro from Fonix Corporation is another technique of this type. *Word-level* techniques, such as Cirrin (Mankoff and Aboud 1998) and Quikwriting (Perlin 1998), allow the input of multiple characters with a single stroke. For example, in Cirrin, a stroke begins in the central area of a circle and moves outward to the first character, then back to the center, then out to the second character, and so on (Figure 9.4).

Another interesting word-level technique, incorporating dynamic rearrangement of letters, is Dasher (Ward et al. 2002). In this technique, the user moves the pen to the first letter in the word or phrase, choosing from all possible letters oriented in a vertical menu. When the first letter has been chosen, the software computes the probability that each letter is the next letter in the word or phrase. The most probable letters appear near the pen and have a larger area than other letters (Figure 9.5). The user continues to move the pen to successive letters until he is finished.

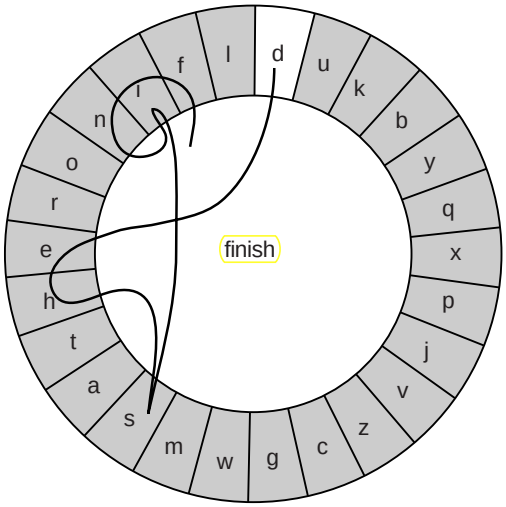

Figure 9.4 *Layout of the Cirrin soft keyboard for pen-based input. © 1998 ACM; reprinted by permission)*

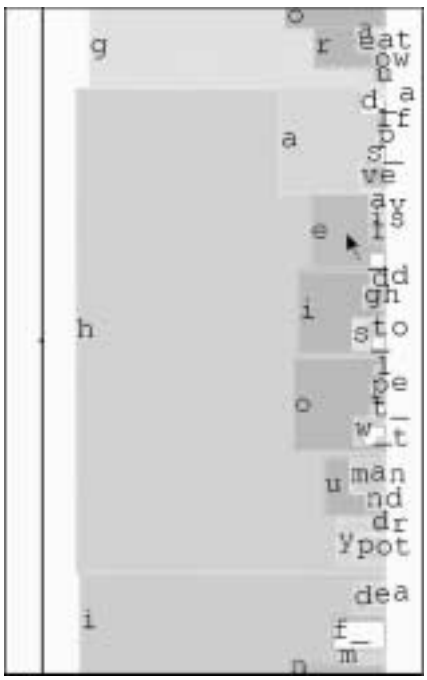

Figure 9.5 *Dasher text input technique. image courtesy of David Ward, Inference Group, University of Cambridge, UK)*

Figure 9.6 *The Virtual Notepad system allowing handwriting in an immersive VE. (Poupyrev, Tomokazu et al. 1998, © 1998 IEEE)*

The Virtual Notepad system (Poupyrev, Tomokazu et al. 1998) used a pen-and-tablet metaphor and simple stroke recognition to provide symbolic input in an immersive VE (Figure 9.6). Handwritten input was also used to command the system. High tracker latency made writing in the original Virtual Notepad system somewhat tedious, but current tracking technology would allow for relatively precise and speedy symbolic input.

Unrecognized Pen Input (Digital Ink)

Another method of symbolic input using pen-based input is to simply draw the actual strokes of the pen, as if the pen was writing with "digital ink." This technique is available on many PDAs as well and allows users to easily and naturally input text, symbols, numbers, drawings, and so on. Of course, the disadvantages of digital ink are that it is only readable by other humans, not by computers, and it is very difficult to edit. Therefore, it may be appropriate for tasks like leaving annotations, but not for tasks like specifying filenames or numeric parameters.

The Virtual Notepad system (Poupyrev, Tomokazu et al. 1998) mentioned above also allowed users to write with digital ink (Figure 9.6). This mode was envisioned for use in medical environments, for example, allowing a physician to make annotations on a patient's x-ray. Annotations in this system could even be edited by using the opposite end of the pen to erase existing strokes.

9.3.3. Gesture-Based Techniques

As we have seen, a great deal of 3D interaction is done with the hands. Thus, it is natural to consider symbolic input techniques that take advantage of hand posture, position, orientation, and motion; in other words, based on *gestures.* Although gesture-based interaction has fallen somewhat out of favor in the 3D interaction community due to difficulty with gesture recognition, calibration of data gloves, and similar drawbacks, it can still be a powerful method of input requiring only the user's hands. We consider three types of gesture-based symbolic input:

- sign language gestures
- numeric gestures
- instantaneous gestures

Sign Language Gestures

One method of symbolic communication based on gestures is already used by millions of people around the world—sign language. Sign language is incredibly descriptive and can allow its users to "speak" very rapidly. Fels and colleagues (1998) have developed the GloveTalk systems to allow signing to be used as a speech synthesizer. Although such a system has not been used in a 3D UI for symbolic input, its use of a data glove as an input device certainly indicates that it might be useful for 3D applications. The main drawbacks of such a technique are that only a small percentage of the population knows sign language and that even for experienced signers, the neural-network-recognition system must be trained.

Numeric Gestures

For numeric entry, gestures offer an obvious interaction technique—the use of fingers to represent numbers (e.g., one index finger raised to represent the number 1). Such gestures are practically universal and can be performed with either one hand or two (in a one-handed interface, the thumb stuck out by itself would represent the number 6). However, we know of no 3D UI research or application that has used this technique.

Instantaneous Gestures

Both of the techniques described above require continuous gesture recognition using a data glove or similar device that continuously reports the angles of the joints on the hand. However, instantaneous devices such as

Pinch Gloves can also provide limited "gestures," especially when a tracker is added to the glove.

The Pinch Keyboard (section 9.3.1) is one example of such a technique, although most of the gestures in that case are meant to emulate the use of a standard keyboard. Certainly, other types of pinch gestures could be used to represent letters, symbols, numbers, editing, or markup. Again, however, there has been little research in this area.

One exception is some unpublished research by Bowman and colleagues that explored various schemes for numeric input using pinch gloves. Ideas included "counting" techniques in which simple gestures on one hand represented the numbers 0 to 4 and simple gestures on the other hand represented the numbers 5 to 9, and "keyboard" techniques in which the hand's position and the pinch gesture were combined to form a virtual numeric keypad layout, similar to the Pinch Keyboard technique. Initial studies indicated that keyboard-based techniques were easier to learn and use, and offered reasonable performance.

A technology called Thumbscript, intended for mobile/phone-based text entry, is another possible instantaneous gesture technique for 3D UIs. Thumbscript is similar to Cirrin and Quikwriting (see section 9.3.2) in that it maps letters to gestures going from one region of the device to another. Instead of pen-based input, however, it uses nine buttons. Users press the button corresponding to the beginning of the gesture and then the button corresponding to the end of the gesture.

9.3.4. Speech-Based Techniques

Finally, we turn to the use of speech for symbolic input in 3D UIs. Speech has a large number of desirable characteristics: it leaves the user's hands free; it utilizes an untapped input modality; it allows the efficient and precise entry of large amounts of text; and it is completely natural and familiar. Nevertheless, speech is rarely used for symbolic input in 3D UIs (or 2D UIs, for that matter). If speech is used at all, it is almost always for system control (entering commands).

The old argument in UI circles was that speech was rarely used because speech recognition systems were slow, inaccurate, and required training, but today's speech recognition technology certainly seems good enough to do the job. Thus, lack of user acceptance must be due to other issues, such as privacy, the perception of bothering others, the awkwardness of speaking to a machine, and so on. Zhai, Hunter, and Smith (2000) also note the interesting findings that "users [find] it 'harder to talk and

think than type and think' and [consider] the keyboard to be more 'natural' than speech for text entry."

Still, because it offers so many potential advantages for 3D UIs, no section on symbolic input techniques would be complete without considering speech input. We also point the reader to the sections on general speech input in Chapter 4 and speech input for system control in Chapter 8. For symbolic input, we can consider three types of speech input:

- single-character speech recognition
- whole-word speech recognition
- unrecognized speech input

Single-Character Speech Recognition

One technique for symbolic input using speech is to have the user utter each character or symbol explicitly. In the case of phrase or sentence composition, this single-character approach would be unwieldy, but in many situations involving the entry of nonword text and symbols (such as filenames), this "spelling" technique makes sense. Furthermore, recognition accuracy may near 100 percent with single-character speech, since the set of possible utterances is so small.

This type of speech input was tested in Bowman, Rhoton, and Pinho's experiment using an idealized "wizard of Oz" recognition system (a human listened to the user's speech and pressed the appropriate key on a keyboard, but users were led to believe that their speech was being recognized by the system). Of the four techniques tested, this speech technique was clearly the most efficient. However, users made significant errors with this technique and had trouble correcting their errors. Furthermore, users perceived their performance to be slower with the speech technique because it was "boring" to spell out the words. Subjects were also observed modifying their speech rate to match their perception of the system's speed.

Whole-Word Speech Recognition

Most common speech recognition software will also recognize a lexicon of words or phrases in addition to single characters. This of course allows speech input to proceed even faster, but perhaps at the cost of increased errors. Such a system would be appropriate when significant amounts of plain text need to be entered (annotations, descriptions, etc.). We are not aware of any 3D UIs that have used whole-word speech recognition for symbolic input.

Unrecognized Speech Input

We can also consider "digital voice," the speech analogue of digital ink. This is speech input that is simply saved as an audio stream rather than interpreted as symbols or words. Like digital ink, unrecognized speech input applies only to symbolic input tasks where another human is the intended recipient of the message. The Virtual Annotation system is an example of a 3D UI that used this technique. It allowed a user (e.g., a teacher) to select an object in a visualization and attach an audio annotation to that object. The annotation was persistent so that later users of the visualization (e.g., students) could listen to the annotation.

9.4. Design Guidelines

We are aware of only three major experiments on user performance in symbolic input tasks in 3D UIs. Bowman and colleagues compared speech, a soft keyboard using a pen and tablet, the Pinch Keyboard, and a chord keyboard in an immersive VE (Bowman, Rhoton et al. 2002). Figure 9.7

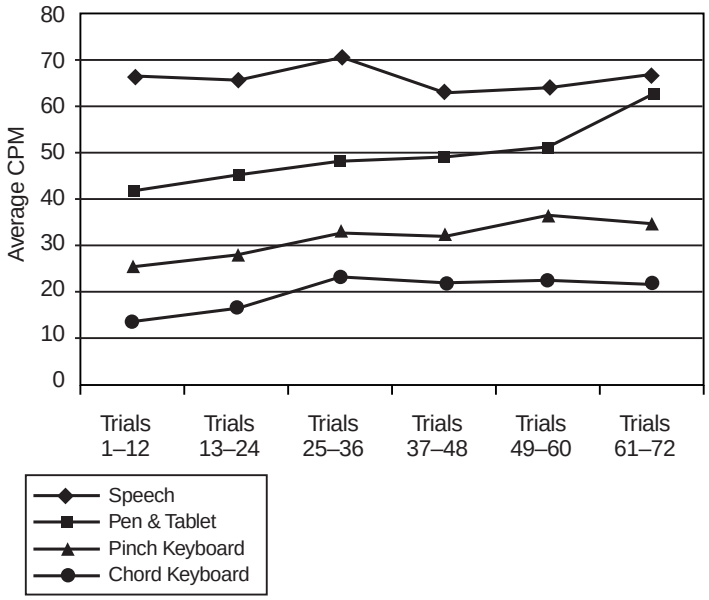

Figure 9.7 *Learning curves for the four techniques in Bowman, Rhoton, and Pinho's (2002) experiment. (Reprinted with permission from Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2002, © 2002 by the Human Factors and Ergonomics Society. All rights reserved.)*

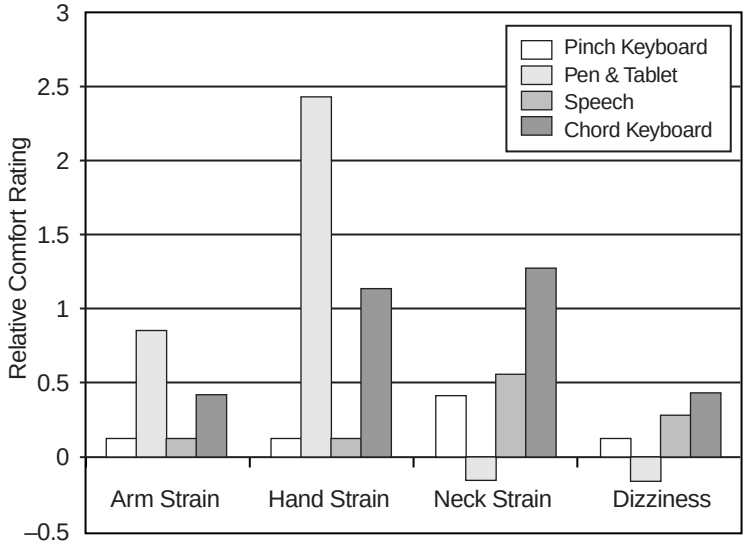

Figure 9.8 *Change in comfort rating from the beginning to the end of the experiment for the four techniques in Bowman, Rhoton, and Pinho's (2002) study. (Reprinted with permission from Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2002, © 2002 by the Human Factors and Ergonomics Society. All rights reserved.)*

shows the average characters per minute achieved by subjects in this study with each of the techniques. Based on pure performance, speech and the soft keyboard technique are clearly superior. Other factors were also important in this study, however. Figure 9.8 shows the average change in user comfort (four categories) over the course of the experiment. The pen-and-tablet technique and the chord keyboard caused discomfort in a large number of users after about 45 minutes of use. Users preferred the pen-and-tablet and Pinch Keyboard techniques, and also found these to be quite natural, since they were based on the QWERTY layout.

The second experiment compared five techniques in a wearable computing setting (Thomas et al. 1998). Techniques included a miniature keyboard mounted on the user's forearm, a soft keyboard, a chord keyboard, speech, and a standard QWERTY keyboard (used as a baseline). Unlike the previous experiment, this study included six lengthy sessions so that subjects would have time to develop expertise with each of the techniques. Table 9.1 shows the performance of each of the techniques in each of the sessions, given in seconds per character (lower numbers are better). As the table indicates, the forearm-mounted keyboard had the best performance of the four novel techniques—even better than speech. The

Table 9.1. *Performance Results for Five Text Input Devices (in seconds per character)*

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6
Forearm	1.03	1.09	0.97	0.87	0.90	0.81
Virtual (soft)	3.29	2.63	2.64	2.27	2.11	2.04
Kordic (chord)	4.75	3.65	3.43	2.63	2.64	2.54
Voice	1.09	1.04	1.03	0.90	1.58	0.90
QWERTY	0.65	0.73	0.44	0.45	0.62	0.45

Source: Thomas, B., S. Tyerman, and K. Grimmer (1998). Evaluation of Text Input Mechanisms for Wearable Computers. *Virtual Reality: Research, Development, and Applications* 3: 187–199.

chord keyboard had the worst performance, but also showed significant gains in speed as users became familiar with the device.

Finally, Osawa, Ren, and Suzuki (2003) examined the use of four mobile-computing text-entry techniques that could be used in a CAVE-like immersive VE. Subjects entered short phrases (in Japanese) using handwriting recognition, digital ink, and a soft keyboard on a PDA, and button input on a mobile phone. They found digital ink (unrecognized handwritten notes) to be the most efficient, followed by button input on the mobile phone, the soft keyboard, and finally recognized handwriting. Subjects preferred the mobile phone technique, however, and found digital ink to be the least preferable method.

Based on these results, we can posit some preliminary guidelines for the use of symbolic input techniques in 3D UIs.

> Use the QWERTY layout if symbolic input will be infrequent or if most users will be novices.

The familiarity of almost all users with the QWERTY layout more than makes up for its deficiencies. Alphabetic layouts, although more understandable at a conceptual level, still force users to search for each character. Other layouts, optimized for pen-based input, two-finger input, and so on may prove much better than QWERTY with extended use, but for "walk-up-and-use" 3D UIs, QWERTY can't be beat.

> Haptic feedback is an important component of keyboard use, so use keyboards with physical buttons if practical. If using virtual keyboards, place the virtual keys on a physical surface.

You don't realize how much you depend on touch when typing until you use a soft keyboard with no haptic cues. The physical outlines of the keys helps users find them without looking; raised dots on certain keys orient users to their finger position on the keyboard; and the passive force feedback helps users understand when a key is actually considered to be pressed. Although soft keyboards can't duplicate these cues, they should at least be grounded on a physical surface rather than floating in the air.

> Don't neglect user comfort.

Avoid heavy devices and those that force the user to hold the hand in a single posture for long periods of time. Consider the other tasks that users will be performing and whether the symbolic input devices will get in the way of these tasks.

> Don't assume that speech will always be the best technique.

Speech can be quite useful for short utterances, annotations, and the like, but it can also lead to error-correction problems, user frustration, and user self-consciousness.

> Consider specialized, nonstandard devices and techniques only if users will be entering symbols very frequently.

Chord keyboards, gesture-recognition techniques, and special-purpose pen-based techniques can all provide reasonably good levels of performance after users have worked with them for some length of time. If your application involves users who can be expected to approach expert performance, these devices and techniques might be appropriate. On the other hand, these techniques will not work when the application is designed for first-time or infrequent users.

> Use digital ink when speed is the most important aspect of usability.

Osawa's (2003) study showed that unrecognized handwritten input could be more efficient than other techniques. Digital ink also allows

users to produce sketches and certain forms of markup. But beware of the limitations of digital ink—it can only be read by other users, not by the system; it is difficult to edit; and it may encourage users to write more quickly, decreasing readability even by other humans.

9.5. Beyond Text and Number Entry

Much more research is needed in this area to enable more complex 3D UIs for future applications. In particular, there has been no work that we are aware of on symbolic input tasks for 3D UIs other than text and number entry. If 3D UIs are going to support complex symbolic input scenarios such as those in section 9.1.2, the challenges of text editing and markup, symbol (e.g., punctuation) entry, and multilanguage support will have to be addressed. While existing work from other types of interfaces will also inform these efforts, there are sure to be unique challenges when the techniques are applied to a 3D UI.

Recommended Reading

A special double issue of the journal *Human-Computer Interaction* focused on "Text Entry for Mobile Computing." The articles in this special issue, while not specifically aimed at symbolic input for 3D UIs, provide good insights into the issues involved in off-the-desktop symbolic input. In particular, the article by MacKenzie and Soukoreff provides an excellent overview.

MacKenzie, I. S., and R. Soukoreff (2002). Text Entry for Mobile Computing: Models and Methods, Theory and Practice. *Human Computer Interaction* 17:147–198.