Connecting the Dots: Simple Text Input in Immersive Environments

Scott Frees[1] Lehigh University*

Rami Khouri[2] Lehigh University*

G. Drew Kessler[3] Sarnoff Corporation

Abstract

We present a novel text input interface for immersive virtual environments called CTD, or "Connect the Dots". The CTD interface is a small virtual panel containing a grid of dots the user can connect to form alphanumeric characters using a hand held stylus. Coupled with a physical paddle or desk for force feedback, the CTD provides an intuitive text input interface similar to using a pen and paper.

CR Categories and Subject Descriptors: I.3.6 [Computer Graphics] Methodology and Techniques - Interaction Techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Virtual reality.]

Additional Keywords: text input

1 Introduction

Text input is an important, yet somewhat overlooked part of interaction in 3D immersive virtual environments. Although it is unlikely anyone will ever do serious word processing while inside an immersive environment, entering small text strings such as filenames or labels requires an effective text input method. Many methods for text input in immersive VR have been proposed, ranging from simple solutions borrowed from the desktop such as hunt and peck virtual keyboards [10] to more complicated approaches requiring specialized hardware such as chord keyboards [6], pinch gloves [1], and chording gloves [8]. Hand-held devices such as PDA's have also been used as text input devises for projection systems [5]. Future text input techniques may utilize advances in gesture [3] and voice recognition.

Bowman et. al. [2] and Thomas et. al. [9] have compared several current text input techniques. Bowman's study compared the virtual keyboard, pinch glove, chord keyboard, and a simplified version of speech. The study suggests that each of the interfaces have their own strengths and weaknesses depending on the given situation. The virtual keyboard seemed to have the best balance between simplicity and performance; however user fatigue was still a problem.

Character recognition is another avenue for advancement. The Virtual Notepad [7] allowed users to write free hand characters, words, and shapes using a hand held stylus. These annotations could be saved as a series of pen strokes and provide sources of information in the virtual world. The Virtual Notepad did not extract semantic (character) data from the drawings however. In the future, this task may be handled by commercial hand writing recognition software and OCR techniques; however the complexity/availability of the software is currently one of several barriers slowing widespread adoption in VR.

Our system is both a simplification and combination of the Virtual Notepad and character recognition reminiscent of the text input interface found in the Palm Pilot. While our interface does not offer the robustness of sophisticated hand writing recognition packages, it does provide an intuitive interface for a user to draw characters that can be accurately recognized by the system.

2 The CTD Interface

The main design goal of the Connect the Dots interface (CTD) was simplicity - from both a user interface and a development standpoint. Our desire was to create a widget that could easily be integrated into our existing VR applications without creating a large footprint in the environment in terms of physical/virtual size or computational resources. To keep the interface as simple and intuitive as possible CTD mimics the pen and paper metaphor.

The "virtual" model of the CTD consists of nine small black spheres protruding out of a thin black backing (shown in Figure 1). The spheres are arranged in a 3X3 grid covering the height and width of the panel, approximately 15 cm x 15 cm in size. The virtual panel is aligned with the top of a desk, podium, or paddle [4] to provide passive force feedback such that when the user rests their stylus on the physical surface the tip of the stylus is slightly above the virtual panel. While the stylus is resting on the panel it can be dragged along the surface to intersect any of the 9 spheres.

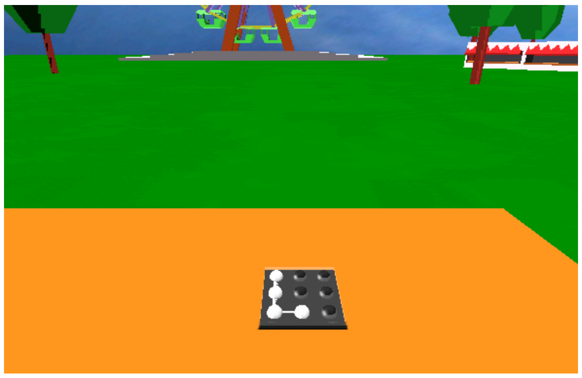


Figure 1. CTD with a virtual desk, the user has drawn the letter 'L'.

In order to enter characters with the CTD the user must rest the stylus on the panel (as they would place a pen on paper), hold down the stylus button, and sequentially intersect the appropriate spheres. Each time a sphere is intersected it turns white to indicate that it is "on". As the user traverses from one sphere (or dot) to another, a white line is drawn between the two (adjacent) dots. The user can use the nine dots and connecting lines to draw out a reasonable representation of an alphanumeric character. Once the user has finished drawing the character they release the stylus button, which tells the CTD to attempt to recognize the character the user has drawn.

* Department of Computer Science and Engineering
[1] sef3@lehigh.edu [2] rhk3@lehigh.edu [3] gkessler@sarnoff.com

The CTD itself has a list of valid character representations, described by which dots are turned on and which lines have been drawn. Note, the order in which dots and lines are activated have no effect on the character description and recognition. Once the user releases the stylus button the CTD compares its current state with its list of character representations and if a match is found a key press event is sent to the application. Note that the CTD does not display the characters entered, it merely sends a key press event to the application. In an application where the user inputs words and sentences, the application itself is responsible for displaying the text, similar to the way any application displays text typed on a keyboard. A few of the valid character representations are shown in Figure 2.

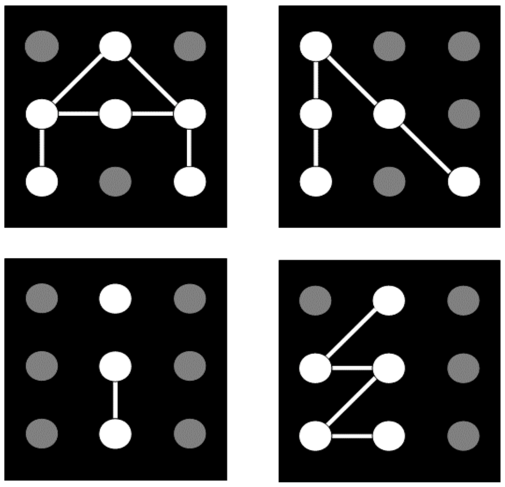


Figure 2. Top left represents 'A'. The top right is an alternative shorthand version of 'A' as well. Bottom left is an 'i' and bottom right is a short-hand representation of 'E'.

2.1 Ergonomic Issues

When using a pen and paper a user can lift the pen away from the paper in order to move to a different position on the paper without making a mark. This is important when drawing disconnected characters such as a lower case 'i', crossing a 't' or simply pausing before the next pen stroke. This functionality is mirrored by the CTD interface. While the user is holding down the stylus button they can always lift the stylus away from the surface of the physical desk (or whatever surface being used for force feedback) to move the stylus without intersecting any dots.

Another usability issue involved in the design of the CTD is its size. Writing small letters with a pen and paper is a generally comfortable and effortless task for most people, involving only the fingers and wrist. Conversely, drawing out large letters requiring elbow or shoulder movement can be fatiguing. Currently, the CTD is 15cm square which is small enough for most people to traverse each of the nine dots using a hand posture similar to that in which they hold a pen, without requiring significant elbow or shoulder movement. While reducing fatigue, shrinking the area of CTD further has proven difficult due to increased error rates. As the dots on the CTD's panel become closer and closer together users have difficulty intersecting the intended dot, especially when trying to write more quickly.

To improve error rates at the CTD's current size we have made adjustments to the arrangement and relative size of the dots. While testing the CTD ourselves and with a few volunteers we noticed that most of the errors occurred when the user was reaching to hit a corner dot. Instead of hitting the target they would instead hit another or miss entirely. This was largely due to the fact that they could not move the tip of the stylus far enough without actually rotating their elbow. Instead of a pure 3x3 grid of dots we have opted to move the corner dots slightly towards the center, preventing the user from needing to reach quite as far to hit the corners. In addition, we have also slightly increased the size of the corner dots to make them easier to hit (Figure 3). Through informal observation and trials we believe these small changes have noticeably improved the usability of the interface.

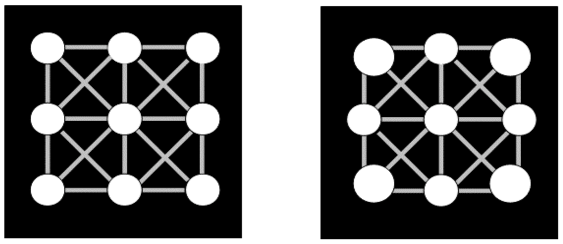


Figure 3. Original layout (left) and new layout to prevent errors at corners (right). For illustration purposes all lines and dots are turned on, revealing the 9 dots and 20 connecting lines.

Of course, no interface can eliminate human error entirely; when the user makes a mistake (intersects the wrong dot) the CTD will not recognize the character drawn or perhaps mistake it for another. This typically happens when the user inadvertently intersects a dot other than the one they intended. Although we have populated the CTD with many different ways of drawing each character, novice users also occasionally attempt to draw a character in way in which the CTD does not recognize.

We have opted *not* to include an "erase" method in the CTD to allow a user to undo an erroneous intersection of a dot. This functionality was omitted this from the interface to keep it as simple as possible, we did not want to introduce modal functionality or additional buttons which might make the CTD more difficult to learn. Instead, the user is instructed to always release the stylus when they have made an error. When the stylus is released the CTD tries to recognize the character as-is, which in turn clears the selected dots and lines. If a character is not recognized the user can simply try again. If a character (other than the one intended) is recognized the user can draw a "backspace" character, which is simply a line across the top row of dots on the panel. The backspace is sent to the application just like any other character and should remove the erroneous character. Although we have not provided a method to erase lines or dots that are turned on, it may be helpful to provide this feature for expert users that can handle the additional complexity.

2.2 Interface Walkthrough

Figure 4 shows the sequence of steps required to draw out the letter 'T'. Starting on the top left of the figure and working across, the user first holds down the stylus button and rests the stylus on the surface of the panel, intersecting the middle dot on the bottom row. While still holding down the stylus button, the user then moves the stylus up through the center dot and up to the middle dot on the top row. Next, the user lifts the stylus off the panel surface (while still holding down the stylus button) and moves to the top left dot. In the lower left figure the user has regained contact with the panel and has intersected the top left dot. Next, the user drags the stylus across the top row to draw the top of the 'T'. Finally, the user releases the stylus button and the letter 'T' is sent to the VR application.

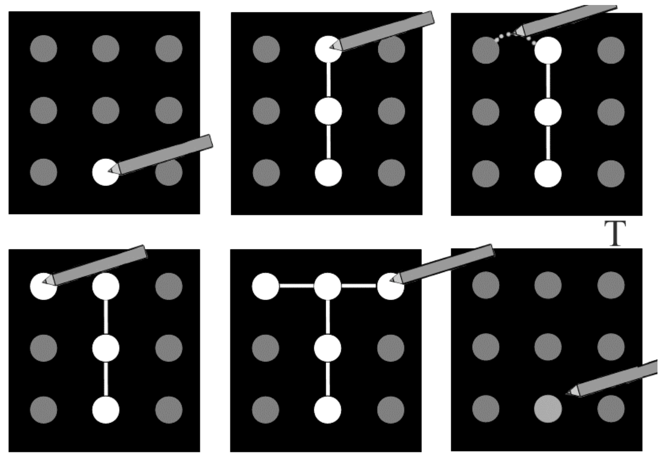


Figure 4. Sequence of interaction steps needed to draw the letter 'T'. Interaction proceeds from left to right, top to bottom. See text for more detail.

2.3 Error prevention and Auto-completion

In contrast to more sophisticated hand writing systems that use true pattern recognition to decipher what character the user has drawn, the CTD only recognizes a discrete set of character representations. These representations are defined by the selected dots and the lines drawn, as shown in Figure 3. Thus, each character representation is simply a set of 29 Boolean values (9 dots, 20 connecting lines).

Although the CTD is capable of recognizing hundreds of different characters (multiple representations of the same character), large lists can still be searched through quite efficiently. This permits the CTD to continuously match its current state to its list while the user is interacting. Continuously checking for valid characters allows for two key enhancements, error prevention and auto completion.

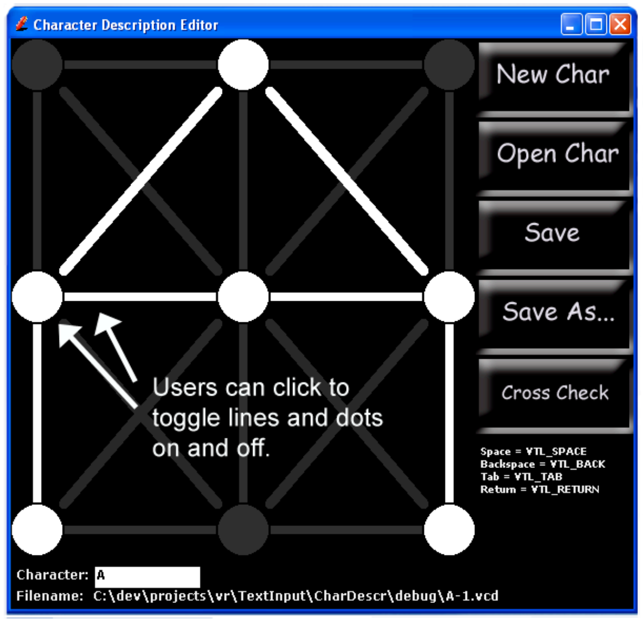
Each time a user intersects one of the dots on the interface the CTD first searches through its list to see if the current character the user is drawing (including the most recently intersected sphere and any resulting lines to be drawn) could possibly be turned into a valid character representation. Since the CTD does not allow a dot or line to be "erased" while drawing, if no character representation in the list includes the particular combination of dots and lines currently selected then the current character cannot be turned into a valid character. Under this circumstance, the CTD will not turn the intersected dot on and will not draw any lines. This prevents errors in two ways: if the user purposely intersected the wrong dot (thinking it would result in the character they were trying to draw) it alerts the user that they have made a conceptual error. More importantly, it blocks inadvertent mistakes allowing the user to simply continue drawing the character as if no error was made.

Auto completion can be achieved utilizing the same technique as error prevention. Once a dot has been selected and the appropriate lines have been drawn the CTD checks to see which character representations the current partial character could be turned into. If the possible character representations all represent the same character (for example, there are multiple ways of writing the letter 'A'), then the CTD will automatically select the remaining dots, draw the remaining lines, and send the application the appropriate character event.

Although these techniques have proven very effective, their usefulness is affected by the size of the set of valid character representations. For example if we include dozens of ways of drawing each alphanumeric character we end up with hundreds of possibilities. The more valid character representations there are the less likely situations where automatic error prevention or automatic completion is possible will occur. This presents a challenge to the designer since maximizing the character representations supported by the CTD decreases the cognitive load on the user since most ways of drawing a character will be supported. Conversely, restricting the set of valid representations will enhance error prevention and auto completion while forcing the user to remember specific ways of drawing each character.

2.4 Character Set Definition

Due to the issues discussed above, the number of character representations (and the representations themselves) a designer may wish to support for a particular application and user population may vary. To allow the designer to easily alter the set of supported character representations we have implemented a desktop application that allows the designer to draw a character representation, specify the character it represents, and save the representation to a single file. This file contains 29 Boolean values (9 dots, 20 lines), which fully describe the character representation. The interface of the desktop application is shown in Figure 5.



Users can click to toggle lines and dots on and off.

Space = VTL_SPACE
Backspace = VTL_BACK
Tab = VTL_TAB
Return = VTL_RETURN

Character: A
Filename: C:\dev\projects\vr\TextInput\CharDescr\debug\A-1.vcd

Figure 5. Screenshot of desktop character description editor.

When creating a set of character representations the designer saves each file to a common directory. The desktop application also provides a way for the designer to check the character representation they are working on against all other character representations in the common directory, which prevents possible collisions between characters (such as identical sets of dots and lines representing a lower case 'b' and the number '6').

When the VR application creates the CTD widget it also provides it with a path to the directory containing the set of character representations it should recognize. The CTD then loads each one of the files in the directory to build its list of character representations. By dynamically loading the character representations from specified folders, designers can easily create different sets of character representations for different applications or users with different preferences.

3 Preliminary Evaluation

At this time we have not conducted any formal usability studies comparing the CTD against other forms of text input for virtual environments, however we have some informal results that shed some light on its effectiveness.

We chose to compare the CTD against one of the most common forms of text input, a virtual keyboard. We have implemented a virtual keyboard that is roughly the same size as a physical keyboard, with characters in the standard QWERTY arrangement. Just as with the CTD, the keyboard is aligned with a physical surface to provide passive force feedback.

For quantitative analysis we created an environment where users were presented a series of five letter (random, but English) words. The users entered these words as quickly as they could. Performance was measured by the average number of characters per second. Error rate was measured by the average number of incorrect letters entered per word. Characters not recognized by the CTD at all were not counted towards this total since no erroneous characters were sent to the application - however this type of error obviously hurt overall performance.

For novices of both interfaces there was a relatively large gap in both performance and error rate between the interfaces. The virtual keyboard proved substantially quicker (around 1 char/sec vs. 0.5 char/sec) and less error prone than the CTD (minimal errors with keyboard, roughly 1 error for every three words with the CTD). For experienced users of the interfaces this gap became smaller, with performance with the CTD approaching 0.8 char/sec and error rate dropping substantially.

Although we clearly need to run more tests comparing the CTD vs. the virtual keyboard, it appears that the keyboard will perform better. This however is not the only method of comparison. In VR, users rarely need to generate significant amounts of text and other qualities of the interface must be examined.

One key issue to look into is the size of the interface. The virtual keyboard has a minimum size somewhere near the size of an actual keyboard, at which point smaller keys (buttons) become difficult to select accurately. The CTD is much smaller than this, allowing it to be placed on smaller and more convenient physical surfaces. The smaller size of the CTD also reduces occlusion and the fatigue issues related to using a large widget or interface.

Another issue somewhat related to size is the interface's expressiveness. When designing a virtual keyboard each new character added to the interface increases its size, since another button is needed. In contrast, the size of the CTD remains constant. A surprisingly large number of characters can be represented using a simple 3x3 grid; our "standard" set includes more than 200 character representations describing over 50 distinct characters including "backspace", "space", and mathematical operators. Our design can be expanded to include symbolic commands (as opposed to alphanumeric characters) that have application specific meaning. We believe this flexibility, along with the ease of modification (through our desktop design application) offers distinct advantages over a virtual keyboard.

4 Future Work

We are confident that the CTD text input interface can be a useful addition to the current set of text input interfaces for VR; however we clearly need to run formal usability tests. It is our desire to compare CTD against the virtual keyboard and perhaps other interfaces to better understand how users can perform with it. Our initial results tell us that performance is lower than that of the virtual keyboard and we have a number of ideas focused on improving those results. Firstly, we would like to investigate increasing the density of dots on the interface to a 5x5 or 7x7 grid. The increased density would allow us to define more variations on each character (more character representations), and make the interface more forgiving, especially to novices. A higher density could also be used to support internationalization. This approach does not come without consequences, as defining a set of character representations reaching into the thousands becomes a long and tedious task. It is our hope that we will be able to develop more efficient mechanisms for defining character sets, perhaps by automatically generating identical representations of the same character that differ only in position on the grid.

Another possibility for increasing usability is to take better advantage of the automatic completion and error prevention techniques described above. To better use these techniques we actually have to *reduce* the supported character set, perhaps limiting it to shorthand styled representations differing from each other enough that the possible characters being drawn can be identified earlier in the interaction.

This trade-off between supporting more representations to reduce cognitive load versus limiting the representations to enhance auto completion and error prevention is not an easily solvable problem. We would like to investigate managing two sets of representations. One set could be tailored towards novices with many different representations of each character. Another set could be maintained for experts who have memorized a small character set and would be better able to take advantage of the automatic completion and error prevention.

References

[1] Bowman, D., Wingrave, C., Campbell, J., & Ly, V. (2001). Using Pinch Gloves for both Natural and Abstract Interaction Techniques in Virtual Environments. *Proc. of HCI International*, New Orleans, Louisiana.

[2] Bowman, D., Rhoton, C., and Pinho, (2002) M. Text Input Techniques for Immersive Virtual Environments: an Empirical Comparison. *Proc. of the Human Factors and Ergonomics Society Annual Meeting*, pp. 2154-2158

[3] Fels, S., & Hinton, G. (1998). Glove-Talkll: A Neural Network Interface which Maps Gestures to Parallel Formant Speech Synthesizer Controls. *IEEE Transactions on Neural Networks*. 9(I), 205-2 12.

[4] Lindeman, R., Sibert, J, Hahn, J., (1999), Towards Usable VR: An Empirical Study of User Interfaces for Immersive Virtual Environments, *Proc. Of the SIGCHI '99*, pp.64-71

[5] Magerkurth, C.; Tandler, P. "Interactive Walls and Handheld Devices - Applications for a Smart Environment". *Collaboration with Interactive Walls and Tables, Workshop at UbiComp'02*, Göteborg, Sweden, September 29, 2002.

[6] Matias, E., MacKenzie, I., & Buxton, W. (1993). Half-QWERTY; A One-handed Keyboard Facilitating Skill Transfer from QWERTY. *Proc. of ACM INTERCHI*, 88-94.

[7] Poupyrev, I., Tomokazu, N., & Weghorst, S. (1998). Virtual Notepad: handwriting in immersive VR. *Proc. of the IEEE Virtual Reality Annual International Symposium*, 126-132.

[8] Rosenberg, R., Slater, M. (1999) A Chording Glove: A Glove-Based Text Input Device, *IEEE Transactions on Systems, Man, and Cybernetics*. Vol 29, No. 2.

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

[10] Zhai, S., Hunter, M., & Smith, B. (2000). The Metropolis Keyboard - an Exploration of Quantitative Techniques for Virtual Keyboard Design. *Proc. of the ACM Symposium on User Interface Software and Technology*, I 19-128.