# Prolix
# A Text-based Participant System for VR

by

Mark Takacs

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science in Engineering

University of Washington

1993

Approved by ______________________________
(Chairperson of Supervisory Committee)

College of Engineering (Inter-Engineering)

August 17, 1993

University of Washington

Abstract

# Prolix
# A Text-based Participant System for VR

by Mark Takacs

Chairperson of Supervisory Committee: Professor Thomas A. Furness III
Engineering (Inter-Engineering)

Virtual Reality has captured the imagination of many people, yet few have experienced it directly due to the currently expensive display and sensor technologies. Those who have experienced virtual worlds often comment on the world's inability to maintain interest with prolonged use. The only thing that remains consistently interesting are other human users. Yet the number of users in a multi-participant world is realistically limited by the great cost of equipping each user with motion tracking sensors, head mounted audio and visual displays, input devices or other interface gear.

This thesis presents a program called Prolix which allows access to the virtual worlds running at the Human Interface Technology Lab (HITLab) without the expensive interface equipment. Prolix assumes only a keyboard and a screen with vt100 capability. Input to the virtual world is accomplished via typed commands and cursor keys, while output appears as narrative text and a simple ASCII-character plan-view of the participant's immediate surroundings. The text interaction style uses the conventions and interface feel of internet Multi User Domains. (MUDs)

An informal pilot study shows that Prolix provides a usable, effective and enjoyable way to communicate with others using the same access interface. Prolix successfully placed a text based person in two VR worlds, one with another immersive user, the other with another Prolix user. However, in the immorsive setting, the Prolix user's graphical display was soon overwhelmed by the volume of information generated by the moving entities and needed to be frequently refreshed.

# TABLE OF CONTENTS

# LIST OF FIGURES

Chapter 1

# SUMMARY

## 1.1 Overview

This chapter is intended to serve as a stand-alone summary of the entire content of the thesis. Each of this chapter's sections corresponds to a whole chapter of the complete thesis.

## 1.2 Related Work and Literature Review

Prolix has several distinct areas: the input of commands, the display of textual output, the graphical display, and the user interface in general. The distilled recommendations from a review of spatial modeling literature, user interface design, previous work, and studies on virtual communities have been summarized in figure 1.1 and figure 1.2.

## 1.3 Prolix User Overview

Prolix is intended to be a participant system that allows entry into virtual worlds traditionally only accessible with fully inclusive participant systems. The users of Prolix interact with the virtual world through the keyboard and cursor keys, and receive graphical display information through the screen, using only ASCII/vt100 capabilities. With this basis in common technology, Prolix should allow many people to experience some form of VR interaction.

The screen is split into two major sections, the graphical display and the text display. Each of which is further divided in two: the graphics display has a plan and side view, while the text display has a separate input and output window.

The graphics window takes up the top half of the screen, with the bottom half devoted to text input and output. The left half of the graphical display shows the

| Prolix Design Issues |
| --- |
| • text content left to designers and participants<br>• atmosphere commands to enhance conversation |
| • conduct a task analysis study<br>• conduct a usability study |
| • atmosphere of a location influenuces behavior there<br>• establish and customize the identity of users<br>• cognative time-sharing occurs<br>• different social norms appear and shape themselves<br>• silence has a different meaning<br>• a social space has commercial appeal |

Figure 1.1: Prolix design guidelines

plan view, while the right side displays an orthographic side view. If an object moves, its icon moves in both windows simultaneously.

The result of text commands issued from the input window are displayed in the output window, as well as output from events in the VR world, including behaviors and communications from other participants and entities.

Navigation through the VR world is accomplished mainly by the use of the cursor keys.

Communication in the VR world is accomplished mainly through the `say` and `tell` commands. The `say` command causes anything you type to be passed out into the VR world for other entities to see and/or process.

Prolix provides a variety of commands that allow you to tailor your perceptions of the world as well as the content of the information you present to the world. Prolix offers the ability to customize the interface and get status information about the current state of the program.

Prolix Programming Guidelines

- use motion instead of static images
- use illustrations and graphics to supplement text
- seperate windows for text output and command input
- display of coordinates
- icon representation with detail lookup
- constant status displays
- seperate input and output windows
- egocentric display
- use digits, rather than graphics, when precise reading required
- similarly orientated objects can be widely spaced without impairing identification

- hierarchical command structure and consistent syntax
- command confirmation
- online help and a complete printed User's Guide
- informative error messages and status displays

- customizable interface for expert users
- locality should be adjustable
- aliases, macros and programmability
- input filtering
- history buffer and session logging

- real-time reactions
- hardware independent software
- allow creation from within the world
- tightly coupled data movement
- discrete movement model

Figure 1.2: Prolix programming guidelines

## *1.4 Prolix Internals*

The participant system's representation in the VEOS VR database is just another FERN entity. Since it is very costly for the LISP-coded prolix-body.fent to do all the processing, the fent makes calls to custom Prolix C functions that are included in the VEOS library.

In general, a fent is controlled by "methods", "react procedures", and "persist procedures". React procedures, known as react procs, are called in response to a change in a part of the VR database that the entity has a registered interest in. Methods are predefined routines in an entity that other entities can trigger upon request. Persist procedures, known as persist procs, are called whenever it becomes an entity's turn to process.

Because the persist procs are part of the entity's turn, they have to be designed to finish as quickly as possible to avoid deadlocking that entity with a long loop. With this in mind, the prolix-body.fent has one persist proc, which does three things, all of which are designed to quickly return a value with as little processing as possible. Figure 1.3 details that persist proc.

```
(defun prolix-body-frame ()                         ;; persist proc
  (let ()
    (if prolix-quit (progn                          ;; exit test
                      (prolix-close)                ;; exit sequence
                      (fern-close)))
    (prolix-keyboard)                               ;; keyboard input
    (if (setq attrs (prolix-comm))                  ;; check changes
          (mapcar 'fern-put.attr attrs))            ;; then post 'em
    (prolix-render)))                               ;; render changes

(fern-persist '(prolix-body-frame))                 ;; persist proc
```

Figure 1.3: Prolix control loop

Before the code in figure 1.3 can be run, the `prolix-init()` function is called once and is responsible for setting up the user's screen and keyboard, and initializing the internal database.

`prolix-close()` is called when the users request it via the `quit` or `<control-c>` command, or when a non-recoverable error occurs. Its responsibility is to close up all the windows, and restore the user's keyboard and terminal to their original state.

If the user hasn't quit, three functions are called in sequential order, starting with `prolix-keyboard()`, then `prolix-comm()`, and ending with `prolix-render()`.

The `prolix-keyboard`() function is a non-blocking keyboard fetch. Its responsibility is to process every single keypress the user makes and perform the associated command. The function doesn't wait for the user to hit any keys, but checks to see if any keys have been hit since the last time it was called. If the user has not hit any keys, the function immediately returns. If there was keyboard activity, `prolix-keyboard()` updates the internal database and returns.

The `prolix-comm()` function controls communication between Prolix's internally maintained database and the VEOS virtual world database. When the function executes, it picks up all the attribute-value pairs that have been prepaired for exporting and returns them in a LISP list. If there is anything in this list, the `fern-put.attr` call posts these new attribute-value pairs to the rest of the VEOS VR database.

The `prolix-render()` function is responsible for rendering the graphical display. The function consists of a single pass through Prolix's internal entity database, checking each entity to see if it should be drawn. Since Prolix uses the `curses` screen optimization library functions, only changes in the graphical display are drawn unless the user requests otherwise.

Two other function groups exist, a `prolix-incoming-()` and an `outgoing-()`. The `prolix-incoming-()` functions provide the communication bridge from the FERN VR world to the screen based interface. They are responsible for conversions from LISP to C data structures and then calling appropriate display and/or database update functions. The `outgoing-()` functions are responsible for displaying a confirmation message to the screen after a command has been issued and calling the fuctions that prepare a message for export to the FERN database.

Prolix accepts most of its command input through typed commands. I used the programming utilities `lex` and `yacc` to generate a lexer and parser, respectively. The lexer assists by forming raw input into meaningful chunks, called "tokens". The parser's job is to recognize valid token streams. The parser passes a string of characters to the lexer, which breaks the string into tokens and returns these tokens to the parser. The parser then compares them to a list of rules for forming valid commands. If the command is valid, the parser calls the appropriate command (usually an `outgoing-()` function), otherwise an informative error message is displayed.

## *1.5 Informal Pilot Study*

The study was designed to take place with multiple subjects using the different interface techniques. The study was conducted to get a quick approximation of the usefulness of Prolix, point out successful features and potential problem areas.

In the first phase of the study, an inclusive subject and a Prolix subject would be placed in a VR world where they can play "tag". In the phase two "chat" study, the inclusive user would switch to a Prolix interface for the purpose of testing the interaction when multiple people use the text interface in the same VR world.

The informal pilot study was conducted with four goals in mind. First of all, I wanted to verify if the interface was usable. Secondly, I wanted to see if the interface was effective at conveying the substance of what went on in the VR world. As a third point, I wanted to see if the text interface uniquely offered anything that the immersive interface didn't. Finally, I wanted comments from people who hadn't been a part of the development effort.

The subjects chosen for the informal pilot study were chosen either for their availability to participate, past experience with text-based interfaces or an expressed interest in the project.

The interface to the tag world was only partially successful. The text based notifications and information look-ups worked as planned, but the graphical display was overwhelmed by the immersive environment. It rapidly became too cluttered, though it worked well for a single subject, many stationary entities, or when the other subject was a Prolix user as well.

The observations during the chat world indicate the Prolix interface was extremely usable and effective in establishing communication between the subjects, as well as effectively conveying information about the VR world.

### 1.6 Applications

Prolix was designed to provide an alternate means to experience VR. Requiring nothing more than a keyboard for input, and the venerable vt100 emulation for simple ASCII only graphic and text output, Prolix should be an accessible alternative to expensive traditional VR interfaces.

Once a participant has connected to a virtual world, the most interesting entities in that VR are likely to be other participants, who you will undoubtedly want to converse with. In the text based interface, communication is provided by typing what you wish to communicate, and this message is propagated out to all the other interested parties.

Some of the worlds accessed can certainly be expected to exist solely for their entertainment value. The entertainment value could come from the intrinsic behaviors encouraged by the particular VR or the social interplay among the participants.

Computer Supported Cooperative Work (CSCW) systems are defined as providing support for a common goal or task, and providing an interface to a shared environment. VR is by its very nature a shared environment. The current goals and tasks commonly include exploration and visualization, with a start being made towards truly interactive artwork. Prolix provides explicit support for common VR goals and tasks at the HITLab.

Since Prolix is text based, there is the potential for online access to an editor, giving a participant the power to shape the world from within.

### 1.7 Future Directions

There is room for Prolix to expand functionally, as well as more elegant ways of programming.

Prolix must function as a fully operational participant system, dealing with renderers, sensors and data management. A conversion to Mercury 2.0 protocols (the HITLab's participant system) would vastly reduce the amount of code required for a text based system. The same functionality of Prolix could be reproduced by writing two drivers, a keyboard sensor to gather and process key hits and a vt100 renderer to show the output. These two drivers would then be added to the library of available devices for the Mercury user to choose from.

Prolix's graphical display suffered from "ghost" images of entities under certain situations, requiring the subjects to refresh the screen frequently. The ghosts appeared when many entities were moving, and were especially prevalent in the immmersive environment. This cluttered the display and obscured any real motion. A solution to this graphical display difficulty must be addressed in any future releases of Prolix.

Prolix uses a discrete movement model where each curser press moves you one grid space in that direction. This also means that you are only moving when you are actively pressing the keys. An alternate model is a velocity model, which produces an illusion of smooth movement, or a "thrust-and-rotate" system.

Currently, Prolix represents any object as a uniformly sized character icon despite the object's real size. A more accurate way to represent objects would be to take some measure of their shape, perhaps by an interpretation of the polygon-graphical description file.

The HITLab's current world development is driven from a LISP front-end where designers can temporarily pause execution, issue LISP commands and resume execution. Having this same flexibility to put the world on "pause" and issue LISP commands would be beneficial within Prolix as well, perhaps with the implementation of a command that left users at the natural LISP/VEOS prompt.

The ability to customize the command structure is appreciated by the expert user, just as the logically grouped command hierarchy is appreciated as the user learns. With this in mind, Prolix could be expanded to include macros and aliases.

Allowing users access to the datafiles representing the entities allows anyone to create entities from within the interface itself. Prolix could adopt a protocol that would allow for the creation and modification of FERN entities and the controlling LISP code.

If Prolix were to be structured as a client that connected to a remote VR world through some protocol, rather than each user running their own copy of the interface system, many more people would have access to VR.

## *1.8 Conclusion*

Prolix provides VR access through one of the most common terminal emulation techniques, and uses only a basic screen and keyboard. No exotic or expensive senser or display devices are required.

Prolix serves as an effective text based communication system for VR worlds. It can better serve as an effective interface to other immersive VR world if the graphical display problems are addressed.

If Prolix is to be truly successful as an alternate interface and tool for social interaction within VR vpaces, the interface must be made available to other institutes, rather than existing specifically at the HITLab.

Prolix and other alternate ways of empowering large numbers of people with the chance to experience VR are beginnig to be developed. As this happens, the content of the worlds will take on much more importance. VR world designers will have to concentrate on providing interesting things for people to do *together* in their worlds, rather than simply constructing visually engaging worlds.

Appendix A

# SOURCE CODE AVAILABILITY

As the source code for the Prolix library additions to VEOS contains more than 7000 lines of code (courtesy of `wc -l *.c *.h *.l *.y`), it would have been impractical to include it here. Tho source code will be available at the HITLab's forthcoming anonymous ftp site (probably `ftp.hitl.washington.edu`) or directly through me.

I would welcome any comments or information requests. I can be reached at the address below until at least December 1993.

Mark Takacs
University of Washington
c/o Human Interface Technology Lab FJ-15
Seattle, WA 98195
tak@hitl.washington.edu
mtakacs@fry.halcyon.com