Scrybe: A Tablet Interface for Virtual Environments

Roger Hoang Joshua Hegie Frederick C. Harris, Jr.

Department of Computer Science and Engineering
University of Nevada
Reno, NV 89557
{hoangr,jhegie,Fred.Harris}@cse.unr.edu

Abstract

Virtual reality (VR) technology has the potential to provide unique perspectives of data that are not possible with standard desktop hardware. The tracking devices often found with such technology allows users to use more natural gestures to interact with these systems. Unfortunately, the practicality of this input modality diminishes as the set of user-controllable options expands. Two-dimensional graphical user interfaces (GUI) are advantageous in this case for both familiarity and organizational reasons. Their application, however, in virtual environments can present difficulties in terms of usability to the user when the interface gives no tactile feedback and floats freely in the virtual world, requiring further interaction to adjust it to a reasonable orientation to facilitate interaction. The ubiquity of touchscreen devices today suggests a potentially inexpensive solution to these problems by providing a tactile handheld interface. We present Scrybe, a tablet interface for virtual reality applications, as a solution to this problem. We describe modifications to a virtual reality library to facilitate the addition of 2D interfaces in two different ways and discuss their application to two touchscreen interfaces: a tablet PC and an Android smartphone.

Keywords: Tablet, Virtual Reality, Human Computer Interaction

1 Introduction

Virtual reality presents the opportunity for users to interact with virtual worlds and data in ways that standard desktop computers cannot provide. Wand devices such as the Nintendo Wii Remote [13] shown in Figure 1 and pinch gloves [10] allow the user to interact with systems by either pointing or gesturing. However, these sorts of input modalities are ill-suited for some tasks. Pointing in 3D space can be imprecise, and the numerous degrees of freedom provided by gloves renders them unwieldy, failing to recognize gestures or accidentally triggering others. For precise selection and execution of commands, the common two-dimensional menu proves effective [1].

Presenting a 2D menu in 3D space has its own difficulties. A menu that floats in the virtual world is bound to be lost by the user as he or she moves about. Alternatively, it can become unreadable or difficult to manipulate if oriented at a bad angle. These problems can be resolved by always rendering the menu in the user's field of view, though doing this results in the menu occluding other objects in the scene. This distraction can be reduced somewhat by minimizing the menu when not needed [1].

Figure 1: The Nintendo Wii Remote, an example gesture and pointing device.

Another problem with these menus is that interaction with them requires either pointing or touching, both of which is imprecise when the additional third dimension is involved. The latter can be particularly problematic with no tactile feedback, leaving the user to guess at how far he or she must reach out to activate a GUI element.

Rendering these menus onto a physical handheld touchscreen device offers a solution to these problems. The user is capable of orienting the screen to a managable position, and the physical surface provides tactile feedback.

Much work has been done on applying physical tablets to enhance user interaction in a virtual environment, though many of these works used a mock surface such as a tracked clipboard and a tracked pen, rendering a virtual interface on top of them using head-mounted displays (HMD). Poupyrev *et al.* did exactly this, allowing users to write notes on a virtual notepad in a virtual emergency

room. Using handwriting recognition, users were also able to enter text commands [6], another input modality that is difficult for virtual environments. Stoakley *et al.* introduced a world manipulation metaphor called World in Miniature (WIM), where users can interact with the virtual world by interacting with a miniature version in their hands. The WIM would be bound to the user's hand as long as the user maintained a grasping gesture; however, without haptic feedback, rotation was found to be difficult. As such, the authors introduced a physical clipboard prop [9]. For their 3D CAD application, Sachs *et al.* used a palette attached to a positional tracker to allow the user to change their view by rotating the palette [7].

Physical tablets are not without drawbacks. Bowman and Wingrave found that while interaction with them can be efficient, holding them can result in arm and hand fatigue in comparison to a glove interface [2]. An opaque mock tablet with an interface rendered through an HMD cannot be used in large screen display environments such as the CAVE^{TM}[3], where rear-projection means that the tablet would obscure whatever was to be rendered on it. Watsen *et al.* present a way to resolve this problem by using Palm Pilot devices to render applets that could manipulate the virtual world [12].

We present Scrybe, a set of augmentations to a virtual reality toolkit that allow a variety of tablet interfaces to be used for VR applications. Our modifications introduce flexibility to the system in terms of how much control over the GUI the application programmer desires. Our system also harnesses a tracker on the tablet device to convert the device itself into a display window into the virtual world, allowing users to either view the world outside of the primary displays or view the world through certain aesthetic or informational filters.

The remainder of this paper is structured as follows. Section 2 discusses the software systems and modifications to it; Section 3 presents the hardware used. Section 4 offers closing thoughts while Section 5 provides avenues for future work.

2 Design

2.1 Problem Background

The problem of how to interact with virtual reality programs, since the virtual world can easily be larger than the physical space provided. There are a number of different input devices that can be tracked in real time, to determine their position and orientation in space, as well as button presses. This does not mitigate the problem of helping users easily interact with program elements that are not best described with a single button press (*i.e.* changing a percentage from 50 to 100).

2.2 Software Design

The goal of Scrybe is to provide tablet interaction to the user for virtual reality applications. It allows the application programmer to do this in two different ways: by allowing the programmer to manage a custom built GUI and by allowing the programmer to register modifiable parameters for which the tablet determines how to best present to the user. This section details how a virtual reality library, Hydra [4], was altered to support these mechanisms. We then discuss two tablet applications created to communicate with any application built with Hydra that exemplify each of these techniques.

2.2.1 Hydra

Virtual reality systems vary widely in terms of hardware. As such, a virtual reality toolkit is generally employed to hide these details from the programmer, allowing their applications to run on a variety of hardware configurations. The toolkit harnessed for Scrybe is a library called Hydra [4]. At its core, Hydra is responsible for providing the programmer with an abstract interface to query for user input and potentially synchronizing this information across a cluster. To provide feedback back to the user, the programmer specifies callbacks for certain rendering modalities he/she choose to support. If the hardware supports a particular modality, Hydra looks for the configured callback and invokes it; otherwise, it simply ignores that aspect of rendering. For example, a programmer can specify rendering methods for OpenGL, DirectX, and OpenAL in the same program, but only the OpenGL and OpenAL methods would be used in a Linux configuration.

Hydra achieves this modularity through a plugin mechanism. At runtime, plugins register services they provide to library. Hydra then examines a configuration file to determine which services the user wishes to invoke, such as a stereoscopic X11 OpenGL window, and finds the appropriate plugin to instantiate such a service. Inputs are treated the same way. If the user wishes to use a particular input device, an appropriate plugin must be provided. This plugin would then convert inputs from this device into a set of virtual buttons, valuators, and tracker objects for the application to read. While plugins expand the ways in which the user is able to interact with the system, extensions provide the application programmer and the plugin programmer with more functionality, such as a graphics system-agnostic way to manipulate the rendering matrices.

To provide the first interaction method for Scrybe, allowing the programmer to manage a custom built GUI system, a single plugin called *GLStreamer* was created. The objective of the GLStreamer object is to stream a video feed to a remote device as well as to receive and redirect user input on the remote device back to the application. The decision to stream images rather than having the remote device render them is motivated by the observation that most portable devices have much lower rendering and computing capabilities than their immobile counterparts. Thus, they would be not only incapable of rendering more graphically complex virtual environments but also incapable of simulating computational expensive applications. Instead, it was deemed more efficient to

have a machine render the necessary images and send them to clients; as such, at run-time, the GLStreamer waits for incoming connections from remote clients. Upon connection, the client sends the plugin information regarding the type of rendering the client needs. The plugin then renders images accordingly to an offscreen render buffer and sends them to the client as it requests them, providing a built-in throttling mechanism.

While streaming raw images was acceptable at lower resolutions, the amount of data increased rapidly as the resolution increased, resulting in the system becoming bandwidth limited. To ameliorate this problem, images are compressed into JPEGs before being streamed. The client is able to dynamically alter the compression quality of the images to adjust for available bandwidth.

Using the GLStreamer object alone, the programmer is expected to both render the GUI and handle all interactions with it. He or she does accomplishes the first task by specifying a render callback that draws both the image the user should see along with the interface over it. The second task is accomplished by specifying another callback that handles mouse clicks and movement. When the user interacts with the image on the client side by clicking, the clicks are sent back to the GLStreamer object which then invokes this callback.

Allowing the programmer to maintain the GUI can be viewed as an unnecessary burden, particularly if the GUI must be made flexible enough to adapt to different screen sizes and resolutions. For this reason, a second solution was proposed that moves the burden to the client application. In such a scheme, the programmer simply exposes certain variables to the client application, which supplies some method for the user to alter them. This functionality was produced with the addition of an extension and a plugin to Hydra.

The extension, dubbed *Parameters*, allows the programmer to specify the type, name, and limitations of a parameter. For example, one could specify a double precision floating point parameter named "Simulation Speed" that is constrained to a value between 1.0 and 1000.0. All maintenance responsibilities, including cluster synchronization, for these parameters is relinquished to the Parameters extension. The server application can then query for changes to the parameters and change its behavior accordingly. Alternatively, the server application can alter the values of these parameters itself; as extension functionality is also exposed to plugins, this can be useful for the programmer that wishes to present read-only information to the user.

The exposure of parameters to the end user is executed with the implementation of plugins. This exposure could come in the form of webpage, a Flash application, or whatever method a plugin developer deems useful. For our tablet applications, a *ParameterServer* plugin was developed. The ParameterServer behaves similarly to the GLStreamer, waiting for remote clients to connect to it. At this point, the plugin sends all information regarding the current state of the Parameters extension to the client. The client application then handles presenting this information to the user. Modifications of these parameters are sent back from the client to the ParameterServer, which then notifies the Parameter extension of the changes. In the reverse direction, if the ParameterServer detects a change to the Parameter extension due to the programmer application, another connected client, or even another plugin, the ParameterServer broadcasts these alterations to all listening clients.

2.2.2 Tablet PC Application

To illustrate the client for the first interaction mechanism, a Qt application, shown in Figure 2 for Linux desktops and tablets was developed. Qt was selected for its ability to quickly generate a static user interface, which is all that is necessary for this method. The static interface allows the user to input the server and rendering properties before connecting to a remote GLStreamer object. It then receives images from the GLStreamer which should contain the dynamic programmer-controlled GUI in them. A slider allows the user to alter the JPEG compression ratio according to his or her tolerances for image quality over interactivity. When the user moves or clicks on the image, information about the position and state of the input device are transmitted back to the GLStreamer object which then informs the application. Figure 3 shows the flow of this arrangement.

2.2.3 Android Application

For the second interaction technique, an Android application for mobile phones was created. The information flow of this setup can be seen in Figure 4. The simplicity of programmatically generating some arbitrary GUI made the Android platform an attractive option. The user can still optionally connect to a GLStreamer to interact in a manner similar to the Tablet application or to simply have a view into the virtual world. Alternatively, the user can connect to a remote ParameterServer object. When

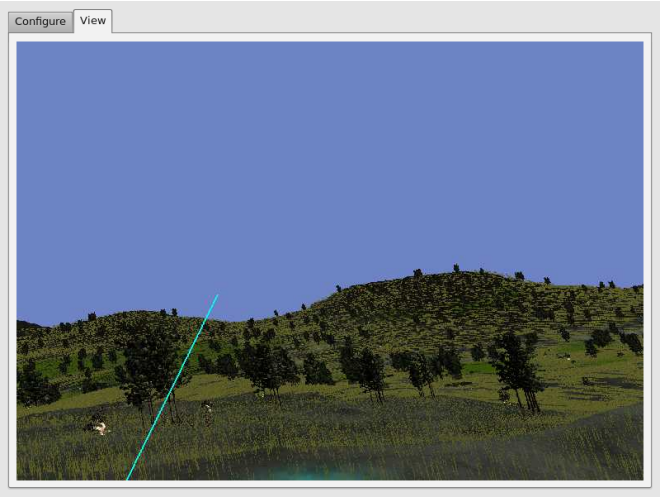

Figure 2: The Scrybe viewing window.

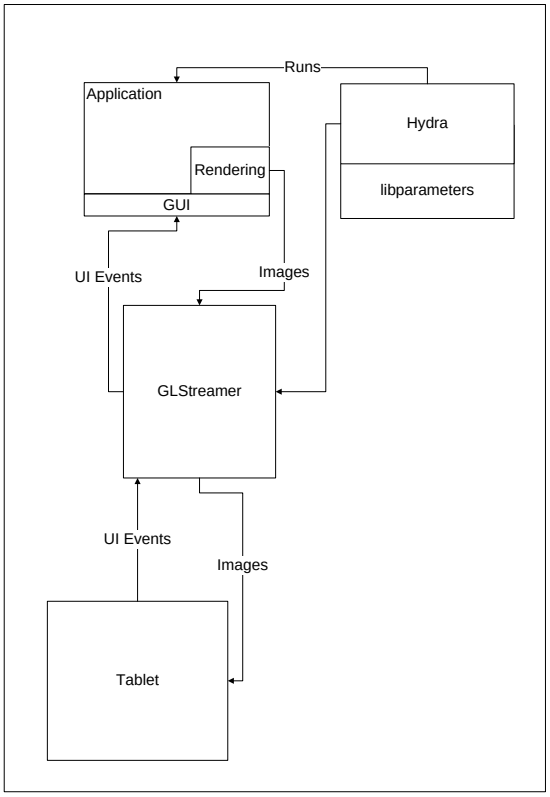

Figure 3: Example Scrybe setup when used with a programmer-maintained GUI.

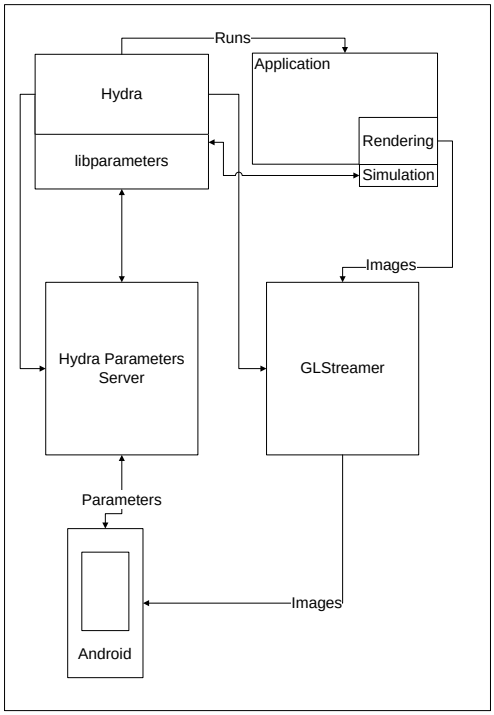

Figure 4: Example Scrybe setup when user-accessible variables are handled by the Parameters extension to Hydra.

connected, the application downloads all parameters from the server and constructs an appropriate list of parameters. Boolean parameters are presented as checkboxes while numerical values are presented with sliders and/or input boxes depending on the range and necessary precision. String parameters are displayed with text boxes. Any alterations to these parameters are sent back to the ParameterServer; meanwhile, a separate thread listens for changes received from the same server.

3 Hardware Design

This project utilized many different hardware components. Foremost in the hardware is a tablet device. A tracking system was also utilized in order to know where in space the tablet is. The last of the major hardware components is the immersive display, which for this project is a singe large screen.

3.1 Tablet

There are currently two variations to the hardware tablet in use, both connected to the sever via a dedicated WiFi 802.11G connection.

3.1.1 Tablet PC

The original idea for this project was to stream to a tablet PC running Linux. The tablet that was initially used used was a Motion Computing LE1700. Though not particularly powerful in terms of the hardware, it is well-suited to what was needed for this project. All that was required of the tablet was that it be able to run Linux Ubuntu 10.04 as of this writing and allow for mouse tracking. This tablet has a stylus which is used as a mouse for both position and for generating mouse clicks.

3.1.2 Android Phone

As the project progressed the development team decided that it may be beneficial to write the software to function on systems other than the tablet discussed above. Android was chosen as a platform for this because the development kit was easy to access, and the hardware was readily available in the form of phones. The main goal of this addition was to make it simpler for users by allowing them to simply connect their phone to the system and then use it like they would the tablet PC. The largest downside of using a phone is that the screen is

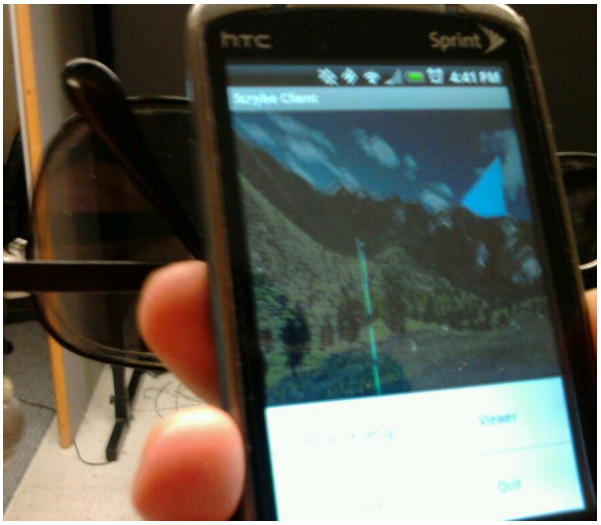

Figure 5: The Android version of the application.

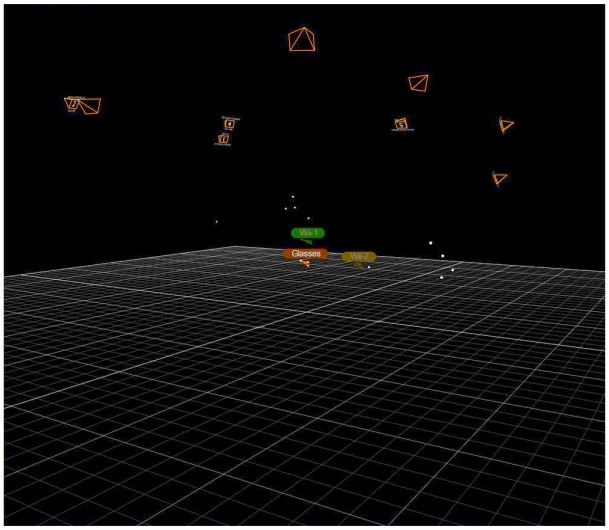

Figure 6: The OptiTrack Tracking Tools.

smaller when compared to a traditional tablet PC, which is not necesarily bad; the image will come across with the same clarity, though selecting specific elements on the screen can be tricky, as everything is now condensed into a considerably smaller area. Another consideration for selecting a phone, or other small scale tablet, is that the framerate that should be streamable should be increased, to account for the fact that the number of pixels needing to be sent is decreased. The Android Scrybe client is pictured in Figure 5.

3.2 Tracking

On top of the standard two-dimensional input devices, using the screen to generate mouse clicks, the tablet's position and orientation are tracked in real-time. This allows the window into the virtual world that the tablet represents to move with the user with a full six degrees of freedom[8]. The tracking system used in this project utilizes ten OptiTrack[5] FLEX:V100 infared cameras which track reflective markers and the proprietary OptiTrack Tracking Tools software, shown in Figure 6. These cameras are accurate to less than a millimeter and are capable of streaming images to the Tracking Tools program at one hundred frames per second. The software is then responsible for maintaining a list of all known trackable objects and updating their positions as they move through the tracked space. This part of the system is essential since it provides a standardized output, in the form of VRPN[11], that can be used to easily update the location and orientation of of the tablet to determine what view of the world to stream over the network.

3.3 Primary Display

The primary immersive display is the backend to the system used for testing. This system is a passive stereo screen that is projected on by two projectors, one for each eye. The light for each eye is polarized differently, so that when both projectors are on, a special pair of glasses allows these images to be reseperated. While not directly part of the system, in the same way that the tablet or tracking, it is required to generate the image that is being streamed over the network.

4 Conclusions

We have presented Scrybe, a system that allows users to interact with a virtual environment using touchscreen devices. To do so, modifications were made to a virtual reality library, allowing the creation of tablet interfaces in two different ways. In the first, the application programmer is given free reign on the aesthetics and organization of the GUI, while in the second, the programmer relinquishes this freedom to the plugin developer and the device developer, who determine the best way to present user-accessible variables. We presented a sample implementation of both techniques for two different device platforms, highlighting the flexibility of the system.

5 Future Work

Many avenues of future work exist for Scrybe. On the one hand, a user study needs to be conducted to measure the effectiveness of tablet interfaces in comparison to gesture interfaces as well as non-tactile two-dimensional interfaces. Exploring other mechanisms to allow for GUI generation would be interesting with respect to programmer usability. The modification of the Hydra library itself to incorporate components of Scrybe would be useful in allowing the user to modify not only the running application but also the configuration of the library.

5.0.1 Acknowledgments

This work was partially supported by by NASA EPSCoR, grant # NSHE 08-51, and Nevada NASA EPSCoR, grants # NSHE 08-52, NSHE 09-41, NSHE 10-69

References

[1] D.A. Bowman and L.F. Hodges. User interface constraints for immersive virtual environment applications. *Graphics, Visualization, and Usability Center Technical Report GIT-GVU-95-26*, 1995.

[2] D.A. Bowman and C.A. Wingrave. Design and evaluation of menu systems for immersive virtual environments. *vr*, page 149, 2001.

[3] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R.V. Kenyon, and J.C. Hart. The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, 1992.

[4] Roger Hoang. Hydra. http://www.cse.unr.edu/hpcvis/hydra/. Accessed May 27, 2010.

[5] Natural Point. OptiTrack Optical Motion Solutions. http://www.naturalpoint.com/optitrack/.

[6] I. Poupyrev, N. Tomokazu, and S. Weghorst. Virtual Notepad: handwriting in immersive VR. In *Proceedings of the Virtual Reality Annual International Symposium*, pages 126–132. Citeseer, 1998.

[7] E. Sachs, A. Roberts, D. Stoops, and C. MIT. 3-Draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications*, 11(6):18–26, 1991.

[8] W.R. Sherman and A.B. Craig. Understanding Virtual Reality-Interface, Application, and Design. *Presence: Teleoperators & Virtual Environments*, 12(4), 2003.

[9] R. Stoakley, M.J. Conway, and R. Pausch. Virtual reality on a WIM: interactive worlds in miniature. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 272. ACM Press/Addison-Wesley Publishing Co., 1995.

[10] Fakespace Systems. Fakespace Pinch Gloves. http://www.i-glassesstore.com/pinglovsys.html.

[11] II Taylor, M. Russell, T.C. Hudson, A. Seeger, H. Weber, J. Juliano, and A.T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*, page 61. ACM, 2001.

[12] K. Watsen, R. Darken, and M. Capps. A handheld computer as an interaction device to a virtual environment. In *Proceedings of the third immersive projection technology workshop*. Citeseer, 1999.

[13] C.A. Wingrave, B. Williamson, P.D. Varcholik, J. Rose, A. Miller, E. Charbonneau, J. Bott, and J.J. LaViola Jr. The Wiimote and Beyond: Spatially Convenient Devices for 3D User Interfaces. *IEEE Computer Graphics and Applications*, 30(2):71–85, 2010.