Tweek: Merging 2D and 3D Interaction in Immersive Environments

Patrick L Hartling, Allen D Bierbaum, Carolina Cruz-Neira
Virtual Reality Applications Center, 2274 Howe Hall Room 1620, Iowa State University
Ames, Iowa 50011-2274, United States

ABSTRACT

Developers of virtual environments (VEs) face an often-difficult problem: users must have some way to interact with the virtual world. The application designers must determine how to map available inputs (button presses, hand gestures, etc.) to actions within the VE. As a result, interaction within a VE is perhaps the most limiting factor for the development of complex virtual reality (VR) applications. For example, interactions with large amounts of data, alphanumeric information, or abstract operations may not map well to current VR interaction methods, which are primarily spatial. Instead, two-dimensional (2D) interaction could be more effective. Current practices often involve the development of customized interfaces for each application. The custom interfaces try to match the capabilities of the available input devices. To address these issues, we have developed a middleware tool called Tweek. Tweek presents users with an extensible 2D Java graphical user interface (GUI) that communicates with VR applications. Using this tool, developers are free to create a GUI that provides extended capabilities for interacting with a VE. This paper covers in detail the design of Tweek and its use with VR Juggler, an open source virtual reality development tool.

Keywords

Virtual Reality, Java, JavaBeans, C++, CORBA

1. INTRODUCTION

Interaction within a virtual world is perhaps the most limiting factor for the development of complex virtual reality (VR) applications. For example, interactions with large amounts of data, alphanumeric information, or abstract operations may not map well to current VR interaction methods, which are primarily spatial. Current practices often involve the development of customized interfaces for each application. The custom interfaces try to match the capabilities of the available input devices. Most of these devices are tailored toward simple three-dimensional (3D) interaction such as grabbing a virtual object or pointing to indicate the direction of travel.

These techniques work well for spatial interaction, but many other types of interactions can be represented more effectively using two-dimensional (2D) interfaces. For example, text annotations, selecting items from lists, and viewing documents work well with existing 2D interfaces in conventional desktop environments. When moving to an immersive 3D environment, users should not be required to abandon the effective presentation methods already available in a standard desktop environment.

In addition to limited interaction in immersive environments, many of the VR interfaces and interactions are closely tied to specific display systems. Although there are applications that demand a particular VR display, most VR applications should be allowed to run on different systems with different displays to take advantage of the available resources or to use the display that is more appropriate to the task at hand.

The need to support 2D interactions in immersive environments combined with the need for a flexible interface that can be used in a variety of VR systems has motivated us to design Tweek. Tweek combines a generalized framework for graphical user interface (GUI) components and a back-end that allows the GUIs to connect to remote components using the Common Object Request Broker Architecture (CORBA) [11]. By using CORBA, the two sides of a network connection can be written in any language and can be operating on two different computing platforms. Currently, Java is the favored language for the GUI, and C++ is the typical choice for the immersive applications. The combination of CORBA, Java, and C++ leads to a wide variety of target platforms and windowing systems, thereby providing the flexibility required for cross-VR-platform interaction methods.

Many new interaction techniques can be explored through this combination of 2D and 3D interfaces. For example, taking advantage of the rapidly evolving palmtop and wireless technologies, we are using a tracked palmtop computer to interact with virtual objects and spaces. This paper covers the design and implementation of Tweek and current use of it with VR Juggler [1][12], an open source virtual reality application development tool. It also describes several applications in the areas of military training, virtual manufacturing, and education in which Tweek has been applied to develop 2D interfaces for palmtop devices.

2. PREVIOUS WORK

Some of the original ideas for Tweek come from our previous work [7][8] on integrating 2D interaction into

immersive environments. This work used a Java-based GUI; however it lacked the CORBA components and a flexible design that could adapt to application needs. Other work in this area [4][6][9] has focused on the effectiveness of using a palmtop in virtual environments. These efforts put little emphasis on the design of a framework that will support user- or application-defined GUIs without requiring customization of the underlying infrastructure. With Tweek, we are building on these earlier works to provide the missing framework.

3. TWEEK MOTIVATION

In this section, we explain the motivation behind Tweek and its novel features, especially as they relate to previous work in the use of 2D GUIs in 3D environments. We begin with the flexibility in choosing a programming language for implementing applications and GUIs. We then explain the importance of GUI portability and what Tweek offers in this area. Finally, we discuss how the Tweek software reuses existing technology, especially that of GUI implementations.

Programming Language Independence

In the field of computing, the state of the art changes constantly, and programming languages are not exempt from these changes. While object-oriented programming is popular today, it may not always be the favored paradigm. Languages that implement paradigms change in popularity as well.

As part of the development of Tweek, we have experimented with multi-language capabilities. The result is an implementation that allows any programming languages to be used for VR application development and for GUI development. The current implementation focuses on Java for the GUI and C++ for the VR application, but this need not always be the case.

Programming language independence in Tweek is realized through the use of CORBA. CORBA itself is a standardized cross-platform, language-independent tool for distributed programming. These characteristics make CORBA well suited for use with Tweek where the VR applications and the GUI are not necessarily written using the same programming language nor do they necessarily execute on the same computer.

GUI Portability

VR applications often must be capable of executing in a variety of configurations, from low-end PCs to expensive, high-end supercomputers. If a GUI will be used with a VR application, it should be capable of moving between systems with the application. Thus, GUI portability becomes an important issue. Some previous work in this area has allowed the GUI to run only in a single computing environment [6]. With Tweek, we have aimed to address such limitations by making the GUI portable from traditional desktop interaction to fully immersive 3D graphics.

With a portable GUI, VR application developers can work with the same interface in a variety of VR system configurations. For example, developers of VR applications often have the ability to develop their environments in what is known as “simulator mode”. Generally, this means using a desktop computer for application design and coding. When running the application, the mouse and keyboard are used to represent the input devices actually available in the full-scale VR system. Thus, the developer can simulate all the actions possible in the VR system using the desktop environment. With this understanding of simulating a VR system on a desktop, we have the motivation for making a GUI input to the virtual world portable between the desktop and the VR system.

Moving beyond the desktop simulation, the Tweek GUI can be brought into CAVE™-like VR systems where large projection screens are used for displaying the 3D graphics [2]. To bring the GUI into the VR system, users may install the Tweek GUI software on a palmtop computer or on a personal digital assistant (PDA). Current wireless technology makes the use of such computing devices in CAVE™-like systems convenient and easy.

When a palmtop computer is not available, the Tweek GUI can make use of 3D windowing tools to run as a 2D representation in an immersive 3D environment. This is because the interface between the user and the virtual environment is separated physically from the virtual reality system and is therefore made portable. This preserves the user’s sense of presence because the interface remains with the user's person as part of the accessible environment, independent of the VR system used.

Reuse of Existing GUI Technology

Current 2D GUI technology is very mature and very well understood. Based on previous efforts to incorporate 2D interaction into immersive 3D spaces, we have found that some interactions are better suited to 2D GUIs than to 3D GUIs or 3D spatial interaction [8]. Thus, the desire to reuse 2D GUI technology has played a major role in designing the Tweek software.

The current Tweek implementation makes use of existing Java-based GUI software libraries. More specifically, we use the popular Java Swing libraries that have been included with all Java Development Kit releases since Version 1.2. As long as a Java virtual machine (JVM) and a windowing system are available, the Tweek GUI software can be executed. The use of Java does impose some restrictions, however, due to JVM availability and

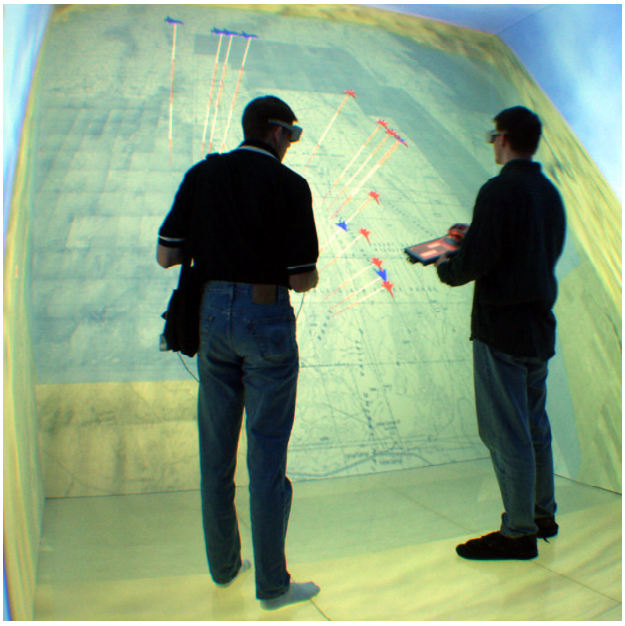

Figure 1 Palmtop in an immersive environment

JVM memory use. For example, most current PDAs have limited memory (both volatile and persistent) and may not have a full Java runtime implementation yet. We feel that the rapidly improving state of PDA technology will lessen the complications imposed by these drawbacks.

By reusing existing Java GUI technology, we have been able to focus on the interaction methods rather than on the implementation of our own GUI software. The language-independence feature of Tweek offers the potential for using other popular cross-platform GUI toolkits such as Qt and GTK+ as well as the use of other operating system-native toolkits.

4. TWEEK DESIGN

The basic design goal of Tweek is to provide a cross-platform, cross-language framework within which VR application programmers can implement 2D GUI interactions with a 3D environment. To achieve this, the Observer design pattern [3] is implemented such that the subject is the VR application and the observer is the GUI. Thus, graphical components within the GUI control act as viewers of state information maintained by the VR application. The state information is entirely user defined. It may, for example, contain the user's position and orientation so that a top-down map can be displayed in the GUI as a navigation aid.

With this basic foundation, users in a virtual environment (VE) can manipulate the 2D interaction on a palmtop computer in exactly the same manner as they manipulate conventional 2D interfaces in a desktop environment. The palmtop provides a physical interface for displaying and using the GUI. It can be integrated into the virtual world simply by bringing the palmtop computer into the space, as can be seen in Figure 1. Input to the GUI is transmitted via a network connection to the immersive 3D application, and the application responds accordingly. In the same manner, the application can communicate information back to the user through the GUI. This enables users, for instance, to add more information about virtual objects; to see specifications of a recently selected object; or to make text annotations about objects using the palmtop input device. By simplifying data input with the 2D device, users can also provide fine-grained specification of numerical values. Object selection and world navigation can be simplified by providing a 2D overview of the environment.

The Tweek software is not limited to display of elements for direct application interaction. Its GUI can load any other GUI component that is needed. For example, a web browser could be run for showing maps or documentation about the immersive world. A scientific visualization application could incorporate a standard database front-end into Tweek so that database manipulations could be visualized instantly.

5. TWEEK IMPLEMENTATION

Tweek is a collection of multiple technologies: C++, Java, JavaBeans™, the Extensible Markup Language (XML), and CORBA. Combined, these allow a Java GUI composed of plug-ins to communicate with a C++ application. We explain the use of these technologies in this section.

Java GUI and JavaBeans™

In order to offer users a highly flexible GUI, Tweek includes a generic Java-based application framework that loads plug-ins dynamically to extend its functionality. These plug-ins may be discovered when the Tweek Java GUI is initialized; they may be loaded from disk after the GUI has already been activated; or they may be downloaded, or "pushed", from the immersive 3D application. The last of these options allows the 3D environment developers to bundle a custom GUI with their application. A user's palmtop can then receive the GUI plug-in automatically when first stepping into the VR system.

The plug-ins loaded by the Java GUI are implemented as JavaBeans™ (Beans) [5]. The Beans fall into one of four categories:

1. Service Beans
2. Viewer Beans
3. Panel Beans
4. Generic Beans

Service Beans encapsulate functionality that may be useful to core elements of the Tweek Java GUI or to dynamically loaded code. The entire interface for a Service Bean must be known when the code using the service is compiled. This is necessary because the using code needs to be able to take full advantage of the service.

Viewer Beans provide a visualization and organization of the loaded graphical Beans. All Viewer Beans must implement a well-known Java interface defined as part of the basic Tweek Java application programmer interface (API). These Beans may be changed at runtime to present users with different views of the same set of graphical Beans. Users of the Tweek Java GUI can write their own Viewer Bean to get a custom view that suits their needs and preferences.

Panel Beans are the key to extension of the Tweek Java GUI. These Beans add components to the generic GUI and thus provide extended functionality. Programmers designing a custom GUI to their VR applications write Panel Beans that make use of existing Java GUI components. All Panel Beans must have a "key" class that derives from the basic Java Swing class `javax.swing.Jcomponent` (or some subclass thereof). This is required so that the key class may be instantiated and added to the GUI layout. Optionally, Panel Bean authors may implement other interfaces defined as part of the basic Tweek GUI API to extend the capabilities of their Beans.

The last category into which a Bean may be classified is the Generic Bean. Nothing is assumed about Generic Beans. This Bean category is provided so that other Beans can do their own dynamic code loading. For example, a Bean that uses a Factory pattern [3] may want to have the "workers" loaded dynamically based on some criteria. In so doing, the functionality of the factory can be changed dynamically.

XML Descriptions

Beans are discovered using XML-based descriptions. The XML description provides all necessary information including paths and external dependencies. An example XML file is shown in Figure 2. This gives an example of a Panel Bean contained in the Java archive `PfControlBean.jar`. Within the archive, there is an entry `vrjtest/PfControl` that is the aforementioned "key" class for the Panel Bean. This class will be instantiated by the Tweek Java GUI and added to the GUI layout.

Observer Pattern

Earlier, we stated that Tweek uses the Observer design pattern [3]. This pattern defines a separation between the state of an object and a view of that state. Within the pattern, there are two classes: *observer* and *subject*. The subject maintains the state; the observer provides a view of the state. A single subject may have multiple observers, each of which is notified when the state of the subject changes. When notified of changes, observers query their subjects to get the latest state information.

```
<?xml version="1.0" encoding="UTF-8"?>
<beanlist>
 <guipanel name="Pf Model Control">
  <file name="PfControlBean.jar"
        entry="vrjtest/PfControl"/>
  <tree path="/"/>
 </guipanel>
</beanlist>
```

Figure 2 Sample XML Bean description

In Tweek, the use of the Observer pattern is implemented with CORBA. CORBA provides the link between the Java GUI and the VR applications. When working with CORBA, all objects are accessed using references. The reference has a well-known interface that can be accessed by any programming language. A language-independent interface to an object is specified using the Interface Definition Language (IDL). IDL is not a programming language, but it includes the basic concepts of *types*, both fundamental and aggregate, and *functions*. Implementations of the interfaces must be written in a programming language such as C++, Java, or Perl.

To simplify the definition of interfaces, Tweek defines two basic interfaces: `tweek::Subject` and `tweek::Observer`. The programmers of VR applications and GUI extend these interfaces to define behavior specific to their applications. Extensions to the subject interface will be used by the VR application to manage state information. Extensions to the observer interface will be used by the Java GUI to visualize and possibly manipulate the state information. Each subject implementation must have a corresponding observer implementation.

To simplify object access further, a third interface, `tweek::SubjectManager`, is defined. An object called the Subject Manager may be created on every node where subjects will be instantiated. Application-specific subjects are registered with the Subject Manager, and nodes that need a reference to a given subject request the reference through the Subject Manager. References are requested using unique, symbolic strings. Thus, the job of the Subject Manager is to hide the details of object registration and reference requesting.

6. CURRENT USES OF TWEEK

Tweek is in use at the Virtual Reality Applications Center at Iowa State University. We use the Tweek Java GUI on palmtop computers to control applications in our projection-based VR systems. For example, we use

Tweek on a palmtop to provide extended input capabilities for a military training application. It includes VCR-like controls that allow playback of recorded training data; six-degree-of-freedom navigation capabilities; detailed entity data readout for individual military units; and visual options for the units and the environment.

This application is written using VR Juggler, an open source, cross-platform framework for the development of VR applications [1]. VR Juggler provides a "virtual platform" that abstracts the details of the VR hardware system. It thus allows the applications to be more portable. When using Tweek with VR Juggler, the GUI interfaces become portable as well.

7. CONCLUSIONS AND FUTURE WORK

The next stage of development for Tweek will include dynamic installation of GUI panels on the palmtop computer. The goal is to allow any user to enter the VE and get the GUI panels automatically. This functionality will be realized by taking advantage of GUI component "pushing", as discussed earlier.

We are interested in the extension of Tweek GUIs into the realm of Java applets, small programs that may be loaded by common World Wide Web browsers. We feel that incorporation into the well-known browser interface could enhance the usability of the 2D GUI, especially for inexperienced users. Related to this, we will investigate encryption and authentication capabilities available with various CORBA implementations to enable secure communication only with authorized parties.

We plan to make use of Tweek GUIs mapped directly into the 3D environment. We plan to experiment with existing work such as 3Dwm [10]. Our goal is to offer an immersive GUI with familiar controls without re-inventing GUI technology.

8. REFERENCES

[1] A. Bierbaum, *VR Juggler: A Virtual Platform for Virtual Reality Application Development*, Master's thesis, Iowa State University, 2000.

[2] C. Cruz-Neira, *Virtual Reality Based on Multiple Projection Screens: The CAVE and Its Applications to Computational Science and Engineering*, Ph. D. dissertation, University of Illinois at Chicago, 1995.

[3] E. Gamma, et. al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, New York, NY, 1995.

[4] G.W. Fitzmaurice, W. Buxton, "The Chameleon: Spatially Aware Palmtop Computers", ACM CHI'94, pp 451–452, 1994.

[5] G. Hamilton (ed.), *JavaBeans™ 1.01 Specification*, Sun Microsystems, Mountain View, CA, 1997.

[6] K. Watsen, R.P. Darken, M.V. Capps, "A Handheld Computer as an Interaction Device to a Virtual Environment", 3rd International Immersive Projection Technology Workshop (IPT 1999), Stuttgart, Germany, May 10–11, 1999.

[7] L.C. Hill, C. Cruz-Neira. "Palmtop Interaction Methods for the Immersive Projection Technology VR Systems", 4th International Workshop on Immersive Projection Technology (IPT 2000), Ames, Iowa, June 19–20, 2000.

[8] L.C. Hill, *Usability of 2D Palmtop Interaction Device in Immersive Virtual Environments*, Master's thesis, Iowa State University, 2000.

[9] M.M. Wloka, E. Greenfield, "The Virtual Tricorder: A Uniform Interface to Virtual Reality", UIST'95 Proceedings, 1995.

[10] N. Elmqvist, *3Dwm: Three-Dimensional User Interfaces Using Fast Constructive Solid Geometry*, Master's thesis, Chalmers University of Technology, Göteborg, Sweden, 2001.

[11] OMG, *The Common Object Request Broker: Architecture and Specification, 2.6 ed.*, Object Management Group, December 2001.

[12] VR Juggler WWW site, http://www.vrjuggler.org/, current April 10, 2002