Graphical Interaction Devices for Distributed Virtual Reality Systems

Marcelo de Paiva Guimarães^{*}
Laboratory of Integrated Systems
University of São Paulo

Bruno Barberi Gnecco[†]
Laboratory of Integrated Systems
University of São Paulo

Marcelo Knörich Zuffo[‡]
Laboratory of Integrated Systems
University of São Paulo

Abstract

This paper explores the possibilities of using portable devices in multiprojection environments, such as CAVEs, Panoramas and Power Walls. We propose and implement a tool to generate graphical interfaces in a straightforward manner. These interfaces are light and can be run on PDAs. The interface application communicates transparently with a graphic cluster, via any underlying network system, which processes the events and maintains the synchrony of the rendered images in real-time. This tool is part of Glass, a library for distributed computing. We present two examples of applications: Cathedral and Celestia.

CR Categories: C.2.4 [Distributed Systems]: I.3.2 [Graphics Systems]: I.3.7 [Virtual Reality]

Keywords: Interaction, real-time, Wireless, PDA

1 Introduction

Input devices are used in Virtual Reality (VR) to let the user interact with the virtual world, providing the feeling of immersion in the environment. Traditionally, VR applications have used specialized devices (gloves, 3-D joysticks and HMD) that are often produced for a relatively small niche market, are extremely expensive, subject to several limitations, such as freedom of movement, and are not easily mastered by inexperienced users.

The integration with PDA (Personal Digital Assistant) devices brings several advantages to VR. Most common PDAs are portable, lightweight and enhanced with fast processors, RAM and ROM capabilities, supporting complex tasks such as the visualization of multimedia data (images, texts, voice, sounds, video/audio streaming), etc. Moreover, they are compatible with wireless standard IEEE 802.11 and with other interfaces, like GPRS [A Sony 2002], Bluetooth [The Official 2004], Serial [The Official 2004] and Infra-red [The Official 2004].

Thanks to these features, PDAs are being used in VR applications, not only to provide a friendly, active participation to the virtual experience, but also to visualize and insert multimedia data and comments both during the virtual experience and after it. The use of PDAs for interaction within multiprojection applications has been investigated in previous works at Tweek [Hartling et al. 2002], JAIVE [Hill and Cruz-Neira 2000] and Virtual Heritage [Benini et al. 2002].

^{*}e-mail: paiva@lsi.usp.br
[†]e-mail: brunobg@lsi.usp.br
[‡]e-mail: mkzuffo@lsi.usp.br

The contribution of the present research project is to simplify the design of graphical interfaces and their interaction with the VR application, freeing the designer from having to know all the details of the individual technologies, and to use PDAs as interaction device. The interfaces generated are simple and intuitive, and can be used even by untrained users to interact with a distributed virtual environment. The proposed solution requires no programming knowledge from the GUI designer. The PDA application communicates transparently with a cluster, via any underlying network system, which processes the events and maintains the synchrony of the rendered images in real-time. Moreover, the interface and its configuration can be changed at run-time. This solution is part of Glass [Guimarães and Gnecco 2004; Cardoso 2003 et al.; Gnecco et al. 2003; Guimarães et al. 2003], a library for the development of distributed applications. It offers key features for distributed VR applications, such as data lock and frame lock synchronization.

This work is based on commodity computers to support the computational demand required by the multiprojection immersive environment. Advances of networks and improvements in personal computer performance have made clusters of low-cost commodity components appealing, allowing their use to solve many problems historically addressed by massively parallel systems [Guimarães and Gnecco 2004; Cardoso 2003 et al.; Gnecco et al. 2003; Guimarães et al. 2003]. These applications use the combined computational power of several resources, while keeping costs low. However, to solve graphic problems, a new category of PC cluster is emerging, the graphics cluster. Graphics clusters differ somewhat from compute clusters in their intent. A computer cluster will take a large problem and break it up into smaller components, distribute the problems to nodes on the cluster that solve each of their assigned tasks, and then synchronize all the information at the back end. In a graphics cluster, the intent is to give multiple views of the same visual data set. Each node of the graphics cluster must have access to the entire data set, then independently determine how much of the data set is visible given its assigned viewing frustum, and render just that part [SGI 2001].

This paper is organized as follows: section 2 is an overview of Glass. Section 3 describes how events are treated by a Glass application. Section 4 presents the PDA interface process generation, section 5 some examples and section 6 the conclusions.

2 Glass

Glass is a new library for distributed computing [Guimarães and Gnecco 2004, Cardoso 2003 et al., Gnecco et al. 2003, Guimarães et al. 2003]. It's a flexible solution, in which all functionality is provided by plug-ins. It was designed from scratch to be a simple, efficient solution for real time distributed computing. This way games, scientific and educational

applications can run easily in a graphic cluster using CAVEs, Panoramas and Power Walls to display the results.

VR applications running in a graphic cluster have specific requirements that include: low delays, fast synchronization, high bandwidth, etc [Guimarães and Gnecco 2004; Cardoso 2003 et al.; Gnecco et al. 2003; Guimarães et. al. 2003; SGI 2001]. Not all available APIs for distributed computing fit these requirements. And those that do are usually too specific: they target multiprojection, for instance, not giving much importance to the underlying computational process — such as physical simulation, real time computation of geometry, etc. They usually provide good resources for graphic rendering (such as scene graphs).

The available frameworks require rewriting the existing code, since it has to be adapted to the new programming paradigm. Legacy code may take a good amount of time and effort to port, and the result is often unsatisfactory. Glass was designed to fulfill this gap, being based on the following principles:

- Extensibility and flexibility: Glass is easy to extend, not requiring API changes or even recompilation. This way it can be kept up-to-date with the latest technologies and be able to solve specific needs of users.
- Portability and interoperability: distributed computing is everyday more heterogeneous. The library can not only run in different architectures (Personal Computers, PDAs) and operating systems (Linux, Irix, Windows), but also interoperate among them.
- Easy to use, fast learning curve: distributed computing APIs tend to be complicated, requiring a lot of time to understand and master. Most libraries provide a huge number of functions, many of them with a high number of arguments. They are hard to learn and remember. Frameworks require a new approach to the programming, and therefore have a steep learning curve. Glass aims to be easy to use, being almost transparent to the user: that's where its name comes from.
- High performance: if an application requires distributed computing, it's because it has a high computational cost.
- Network protocol independency: an abstraction of the underlying network protocol makes Glass protocol independent. Protocols such as TCP, UDP or even more high-level systems such as MPI can be used, by instantiating the base class. This is quite important in a world where applications have hugely different network requirements.
- Reliability and fault tolerance: any distributed computing library must be reliable and fault tolerant. It's unacceptable that failure of a single node crashes the entire computational system. Glass provides fault tolerance automatically.
- Thread safety: many APIs are not thread safe. This forces the user to find workarounds or use other solutions instead of threads. Glass works in thread applications seamlessly.

Glass is completely written in C++. The choice was due to its wide number of compilers and spread of use, good performance, and certain features that would make

programmer's life easier (such as OO and templates). C++ is easy to interface with C, Java and other languages.

A core provides all internal functionality, such as the network system, plug-in management, node management, etc. The developer is completely oblivious to this core.

3 Events in Glass Applications

The feeling of immersion in VR is created by the output devices and by immediate response of user interaction [Zyda 1999]. The treatment of events generated by the input devices depends on the necessary time to process the data and the amount of generated data. Data processing can either be done in a single node or distributed among several nodes. One must guarantee, however, that the data is processed correctly, in a deterministic way. Consistent treatment of input data is of extreme importance in multiprojection applications, because the viewpoint of each rendered image must be accurate to prevent incoherence of the displays. All images must be presented in perfect synchrony (that is, both frame lock and data lock have to be done).

Our initial measurements show that the delay produced resulting from the network communication is less than a tenth of milliseconds, even using TCP, as it is being done currently. This lag is imperceptible to the user, not hindering the interaction in anyway. The responsiveness is at least as good as the one provided by a keyboard.

The Glass Event plug-in is the component responsible for events. Events are queued and propagated asynchronously to all interested nodes. Events are stored internally in a FIFO, and the application reads them as necessary. Optionally, you can set a callback that is called whenever an event arrives.

Figure 1 presents a code snippet showing how to use events. First the event "e" is declared. Events are templates, so they can be used with any data type. Then two events are enqueued (values "12" and "326"). Finally, we check to see if the queue is empty; if not the variable "x" gets the event at the top of the queue. The PDA code is generated automatically by the Graphical Interface Generator (no programming: see section 4.1).

```
//Declare an event variable
Event<int> *e = new Event <int> ("e");
...
//Enqueue data
e->enqueueEvent(12);
//Enqueue data
e->enqueueEvent(326);
...
//Get the event from the top of the queue
while (!e->isEmpty()) {
    int x = e->getEvent();
}
```

Figure 1 – Using events in Glass.

Event variables are useful in a multiprojection environment. They propagate the user's interaction immediately to the interested nodes. Glass also provides functionality to perform frame lock and data lock.

4 PDA user interface

A tool to simplify the creation of GUIs was created. This tool, written in Java, allows the GUI designer to define the background image (gif, jpeg), hot spot areas in it and associate them to events. The user just clicks with the PDA pen on the hot spots to interact with a distributed virtual environment (no training is necessary). When a hot spot area is clicked, the associated event is sent to the interested nodes (section 4.1). Once the GUI is finished, the tool generates a GUI configuration file, and a code skeleton to process the events in the main application (section 4.2).

4.1 Graphical Interface Generator

The graphical interface generator tool consists of a Java application. Java was chosen due to its platform independence.

The tool proposes a simple approach to GUIs. Instead of the usual set of widgets present in most GUI APIs, an image is used. The designer then defines hot spot areas on this image, which can be associated to events of the application. A complex GUI, with alphanumeric fields, menus, etc., reduces the feeling of immersion, since the user has to concentrate on the device instead of on the application.

A screenshot of the tool is shown on Figure 2. One can see on the center the GUI image, which will be shown on the PDA, and the colored hot spots over it. On the right there is a list of the hot spots and the events that are associated with them. The designer uses the buttons on the toolbar to select, add and erase the hot spots. The hot spots can be either a rectangle or an ellipse, and hot spots associated with the same events can overlap. The menus supply resources to create, close, and save the project, and to resize the image area. Image size area present on the Figure 2 is configured to an iPAQ H3970 (240 x 320).

Figure 2 – GUI design tool, showing the Celestia interface.

Each hot spot is associated to an event, which has a number (automatically generated), name, color and type. One event can have more than one hot spot mapped. There are three types of events:

- Simple event: when triggered, the associated event number is sent to the interested nodes, which will process it.

- Exit event: when triggered, the application running on PDA exits, notifying the other nodes.
- Skin event: when triggered, the PDA skin (background image) is changed. An application may have a set of skins, and each skin may have a different set of hot spots. These events are attended locally, but a notification may be propagated to other nodes.

The decision of treating events remotely simplifies the development of the GUI. It's also easier to change the GUI, since it requires no change or recompilation of the application. The GUI designers don't need any programming knowledge.

4.2 PDA code

The graphical interface generator tool creates a data file to be used with the GUI Java application. A skeleton of the event handler on the main application is also generated automatically (in C). This routine will receive the events from the PDA on the main application and process them.

Figure 3 presents a skeleton example. This code can be used as initial base to implement the event handler. When the user clicks on the hot spot "Mercury", for example, this routine will receive the associated value and the switch will handle it.

```
enum Event = {Mercury=1,
               Venus=2
};

switch (event) {
    case Mercury:
        // to be filled by the developer
        break;
    case Venus:
        // to be filled by the developer
        break;
}
```

Figure 3 – Skeleton code of the event handler.

Figure 4 shows the application environment. On the middle is a multiprojection device (CAVE) that is being controlled by the computers. The PDA is considered just another node of the cluster, so it offers bidirectional communication. The PDA application can be the same one that runs on the PDA. Since events are asynchronous, the other cluster nodes do not depend on PDA performance to do the data and frame synchronization. The PDA node can be initialized or shut down at any moment, and the application will keep running. This is useful to have several PDAs running simultaneously — for instance, each PDA may be controlling a different vehicle of a simulator.

Figure 4 – The PDA is a cluster node.

The PDA application uses the Swing Java component. The use of Java allows the application to be executed on many different operational systems (Windows, Linux, Windows CE, IRIX) and platforms.

Figure 5 shows the layers of the generated code. One could integrate this code in more complex applications, using either Java or C/C++. JNI (Java Native Interface) is used in our PDA application to interface with Glass. On top of the operational system is the Java Virtual Machine. The highest layer is our Java Application.

Figure 5 – PDA software layers

5 Examples

All tests were performed on a 6-node cluster of PCs, codenamed POLUX, at the Integrable Systems Laboratory at University of São Paulo. Each node is a Dual Pentium III Xeon 1GHz , 2MB cache, 1GB RAM, Wildcat 4210. The nodes are interconnected by Gigabit Ethernet network. A PDA IPAQ H3970 with a 240 x 320 screen running Linux was used. It is a slim and powerful platform. The communication technology used was IEEE 802.11b High Rate standard wireless LAN (WLAN) interface working at 11 Mb/s because our application was for indoor environment and the costs of the infrastructure were reasonable.

We ported some applications to test the PDA use in VR environment. Figure 6 presents a PDA interface example. Note that the hot spots are invisible.

Figure 6 – PDA running a sample interface.

Figure 7 shows a user using a PDA to navigate in a Celestia [Celestia 2004]. It is a free astronomy application, released under the GNU Public License. We modified its source code to support multiprojection environments and to be controlled by a PDA. This example shows that even a relatively complex code can be easily ported to run in a cluster and controlled with a PDA using our approach.

The Celestia PDA application has a set of skins (Planets, Stars, Moons, Comets and others) which can be changed on run time. For example, if the user wants to navigate on the planets he changes to the Planets skin, if he wants to navigate on the stars, he changes to the Stars skin.

Figure 7 – User navigating in Celestia application using a PDA.

Figure 8 shows a user using a PDA to navigate in a Cathedral.

Figure 8 – User navigating in Cathedral application using a PDA.

Model provided by Marko Dabrovic (mdabrov@rna.hr)

6 Conclusion

PDA's not only improve the user interaction in immersive environments, but can also be used to control the VR system itself, running applications, controlling lights and projectors, etc. It's an efficient way to manipulate large volumes of data, since the user is already familiar with the GUI approach used. Our tests show that even novice users that are not familiar with computers interact in a natural manner with these applications, and in a matter of seconds are navigating in the virtual worlds without conscious effort to control the interface. Unlike all other known input devices, PDA's can provide a visual feedback to the user, adapting its interface instantaneously.

We presented a tool for designing GUIs, which generates code automatically, requiring no knowledge of computer programming from the user. The interface runs on Java, which has the advantage of easy portability to any platform. The routine for treatment of events on the main application is also generated automatically. More importantly, the interface integrates directly with the main application, which is running on a cluster. No extra effort is needed by the developer to support the interface: all input is propagated as asynchronous events, which can be either polled or handled by callbacks on the application. Support for the application is no more difficult than associating the events to the functions that they should perform.

Unlike most input devices, there's no limit to the number of PDA's that can be used simultaneously in an application. Each PDA may be controlling a different vehicle, or the participants may share a GUI and work collectively on a project.

As future work, we plan to enhance GUIs to be multimedia, including animations and sounds. The Glass library will be publicly released soon, as an open source project.

References

- A SONY Ericsson apresenta cartão modem GPRS para PC . 2002. http://www.ericsson.pt/pressroom/other/pressreleases/20020305_0005_SonyEricsson_GC75.pdf.
- BENINI, L. and BONFIGLI, M.E. and CAROLI, L. and FARELLA, E. and RICCÓ, B. 2002. *Palmtop Computers for managing Interaction with Immersive Virtual Heritage*. Proceedings of EUROMEDIA2002, pp. 183-189.
- CARDOSO, A. et al. 2003. *Ambientes Virtuais: Projeto e Implementação*. Sociedade Brasileira de Computação. Porto Alegre.

CELESTIA. 2004. <http://www.shatters.net/celestia/>.

GNECCO, B.B. and GUIMARÃES, M.P. and ZUFFO, M.K. 2003. *Um framework flexível e transparente para computação distribuída de alto desempenho*. Simpósio Brasileiro de Realidade Virtual. Ribeirão Preto.

GUIMARÃES, M. P. and GNECCO, B.B. and CABRAL, M. and SOARES, L., and ZUFFO, M. 2003. *Synchronization and data sharing library for PC clusters*. VR-CLuster'03 - Workshop on Commodity Clusters for Virtual Reality Los Angeles, USA. March 22th-26th.

GUIMARÃES, M. P. and GNECCO, B. B. 2004. *Glass*. <http://www.lsi.usp.br/~paiva/glass/glass.htm>.

HARTLING, P. and BIERBAUM, A. and CRUZ-NEIRA, C. 2002. *Tweek: Merging 2D and 3D Interaction in Immersive Environments*. 6th World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, Florida.

HILL, L., and CRUZ-NEIRA, C. 2000. *Palmtop interaction methods for immersive projection technology systems*. Fourth International Immersive Projection Technology Workshop (IPT 2000).

SGI Graphics ClusterTM: *The Cluster Architecture Challenges, the SGITM Solution*. 2001. Visualization Solutions Development Group. White Paper. <http://www.sgi.com/products/legacy/pdf/architecture.pdf>.

THE OFFICIAL Bluetooth Web site.2004. <http://www.bluetooth.com/>.

ZYDA, M.1999. *Networked Virtual Environments: design and Implementation*. SIGGRAPH series.