VEGI: Virtual Environment GUI Immersion System

Mohammed Elfarargy [1], Magdy Nagi [1,2], Noha Adly [1,2]

[1] Bibliotheca Alexandrina, El Shatby 21526, Alexandria, Egypt
[2] Computer and Systems Engineering Department, Alexandria University, Alexandria, Egypt

ABSTRACT

Virtual Reality (VR) immersive environments are becoming more popular and of less cost, hence, VR labs are becoming a main part in any research that depends on visualization. This introduced the need to port many 3D desktop visualization applications to VR. Porting application GUIs can be a problem since original GUIs are 2D by nature and using them directly can obscure a large area of 3D viewport and spoil the immersive experience. On the other hand, rewriting a 3D GUI can be a time consuming and tedious task. In this work, we introduce a technique to embed 2D GUIs into 3D Virtual Environments (VE). Our approach uses existing 2D GUIs that can be immersed into the VE allowing rapid GUI development for VR applications. It can also be used for porting 3D desktop applications without rewriting the GUI code. Further, it enables embedding many window-based desktop applications into the VE, creating rich VEs where users can work with multiple applications simultaneously.

KEYWORDS: Virtual Reality, Graphical User Interfaces.

INDEX TERMS: I.3.7 [Three-Dimensional Graphics and Realism]: Virtual reality; H.5.2 [User Interfaces]: Graphical user interfaces (GUI) - Interaction styles; H.5.1 [Multimedia Information Systems]: Artificial, augmented, and virtual realities.

1 INTRODUCTION

There are many ways to build a GUI for a 3D VR application. One option is to build a 3D GUI from scratch specifically for a certain application. Many innovative ideas either in the GUI itself or in input devices used were implemented to create 3D GUIs[7]. One downside of this approach is the lack of stable and standard frameworks to build such GUIs, which makes developing them a challenging and time consuming task for developers for each new application. Also, considering this method when porting existing 3D desktop applications to VR environments means that the whole GUI code must be rewritten from scratch which is time-consuming and effort demanding.

Another approach for making GUIs for 3D applications is to embed a 2D GUI inside the 3D application. This usually requires 3 steps:

1. Take a snapshot of the 2D GUI window displayed on the desktop and use it to texture map some 3D object.
2. Send user events from the 3D application to the original GUI.
3. Take another snapshot of the window to reflect changes caused by user interaction.

This technique can be implemented in many ways by utilizing operating system's window APIs (XLIB for Linux, Windows API for MS Windows, etc) or third party Widget Toolkits (WT) that provide the capability of sending user interaction events (clicks, key presses, mouse movements, etc) manually and capturing GUI snapshots. Andujar et al [2] and Larimer [3], used the Qt WT for handling this process in order to make 2D GUIs and then have them immersed inside VR applications. This approach can be ineffective when porting existing applications to VR environments since the original GUI must be created using the same WT; otherwise, the GUI code must be rewritten.

Other approaches [1,4] used XLIB window programming library to insert 2D windows into 3D applications. These works targeted 3D desktop applications rather than VR applications. While the concept is interesting, desktop users can find it difficult to interact with embedded GUIs. Also, it would be easier to interact with the original 2D GUI instead of embedding it in a 3D environment and then interacting with it. Using the same concept with VR applications can be more useful since users usually use only one 3D application that covers the whole display. Putting any 2D windows on an immersive environment would obscure the display area and interfere with immersive experience.

This work describes Virtual Environment GUI Immersion (VEGI) system, which uses the same concepts as in [1] and [4] for time-efficient porting of desktop 3D application to immersive environment. Without any additional coding, VEGI embeds 2D GUIs of interest and lays them on 3D planes and allows interaction with them. Also, VEGI makes it possible for VR application users to use multiple desktop window-based applications simultaneously such as web browsers or calculators while using the main VR application which results in a richer work environment.

2 SYSTEM ARCHITECTURE

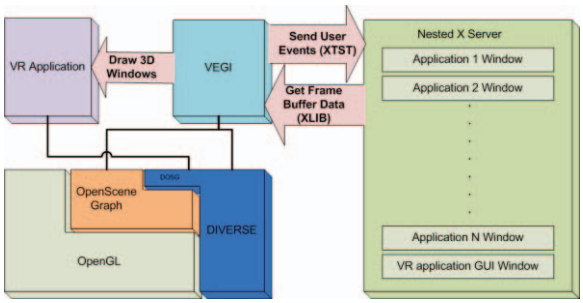

Figure 1. Basic system structure and functionality.

Figure 1 summarizes VEGI architecture. OpenSceneGraph [6] is the main graphics engine used to draw 3D planes, texture them with 2D GUI windows and manipulate them. DIVERSE [5] VR library handles the display of 3D graphics inside the VE. The 2D GUI windows of the application to be ported, or any other window desired inside the VE, are drawn to a nested X server such as Xvfb, Xephyr or Xgl.

VEGI is loaded at run time as a DIVERSE Dynamic Shared Object (DSO) that tracks windows immersed into the VE including those of the main application GUI. A simple configuration file is needed to specify which windows to track and their initial placement in the virtual world.

e-mail: mohammed.elfarargy@bibalex.org
e-mail: magdy.nagi@bibalex.org
e-mail: noha.adly@bibalex.org

IEEE Virtual Reality 2011
19 - 23 March, Singapore
978-1-4577-0038-5/11/$26.00 ©2011 IEEE

3 Implementation

VEGI uses various XLIB routines to capture screenshots of the 2D windows of interest and track new windows created at runtime. Sending user interactions with windows is done using the XTEST extension by using 6 degrees of freedom wand ray-casting to send mouse cursor movements and button clicks.

A major difficulty is keeping track of new windows creation at runtime. The way WTs handle window creation and destruction differs from a WT to another. Taking pop-up windows as an example, it is noticed that GTK+ library does not create a pop-up window until needed. After usage, the pop-up window is not destroyed and is kept unmapped until required again. FLTK, on the other hand, creates pop-ups when needed and destroys them soon after usage and creates them again when needed and so on. Knowledge of such library-specific behaviours is required to handle status changes correctly and hence it is essential to know the WT behind newly created windows. Our prototype currently supports GTK+, FLTK and Qt WTs.

X-server implementation varies among Linux distributions. Needed GUIs are drawn to a nested X-server instead of the host X-Server to guarantee the availability of a fixed set of features and improve portability. Also, nested X-servers will host only the GUIs we want embedded in the VE. This results in a minimal window hierarchy that is easy to traverse. This improves performance compared to host X-servers where hundreds of windows and child controls exist resulting in huge hierarchies.

4 Sample Applications and performance evaluation

Various applications were used to test and evaluate VEGI. GTK+ WT support was tested by embedding a number of popular GNOME applications in the VE. VEGI was also used with a natively developed VR application for statistical data visualization that uses Qt library for GUI. Finally, VEGI helped in making a VR version of VMD software, which uses FLTK library for GUI (Figure 2). The VR version of VMD is an example of how VEGI can be used to easily port a visualization application to immersive environments. VEGI was tested on a cluster of five PCs equipped with Dual-Core 64 bit 3.60 GHz Intel® Xeon® Processors with 4GB of RAM and 128 MB Nvidia® GeForce® FX 4500 cards.

Figure 2. VMD on CAVE system.

Two performance tests were conducted using CAVE system working in stereo rendering mode. Both tests immersed 3D GUIs into a 3D scene of 1,500,000 triangles. The scene rendered at steady 55 frames per second (FPS) without VEGI loaded.

The first test evaluates the impact of the size of the immersed 2D GUI on performance. Larger GUIs mean larger pixel data in the frame buffer. In order to update planes' textures, this portion of the frame buffer must be copied and used to texture-map the plane. The larger data to be copied, the more overhead there is. Square GUI windows ranging from 64x64 pixels to 1024x1024 pixels were used. Results show that VR applications can work with an acceptable frame rate of 30+ fps with VEGI for most common window sizes. Table 1 summarizes these results.

Table 1: Effect of increasing 2D GUI window size on performance.

Window size (pixels)	64x64	128x128	256x256	512x512	1024x1024
Average FPS	52	51	45	36	22

The second test evaluated the impact of the number of immersed 2D GUIs. Table 2 shows that performance dropped by an average of 1.5 fps for each additional window which is acceptable. The VMD application was used with up to 10 GUI planes without any noticeable delay. These results show that VEGI can be integrated with VR applications without affecting the interactivity and performance of the VR application.

Table 2: Overhead of increasing the number of immersed windows.

No. of windows	1	2	3	4	5
Average FPS	53	51	49	48	47

5 Conclusion

We introduced an approach for developing GUIs for VR applications that provides a quick and easy way to port 3D desktop applications into VR domain by embedding their 2D GUIs into the VE. VEGI can also be used to embed any number of window-based applications into the main VR application.

This work can be improved in many ways in the future. One limitation in this work is that it currently supports DIVERSE VR library only. Other VR libraries like *FreeVR* and *VR Juggler* should be supported. Also, support for additional WTs should be added to allow more applications to be ported to VR.

Acknowledgements

The authors wish to thank Ahmed Nawar for his valuable efforts in testing, evaluating and improving VEGI.

References

[1] Alexander Topol, 2000. Immersion of Xwindow applications into a 3D workbench. Conference on Human Factors in Computing Systems (CHI'00) Student Poster, The Hague, The Netherlands, pp. 355–356.

[2] Andujar C. et al, 2006. A cost-effective approach for developing application-control GUIs for virtual environments. Proceedings of the IEEE virtual reality conference (VR 2006), Washington, DC, USA, pp. 45–52.

[3] Daniel Larimer and Doug Bowman, 2003. VEWL: A framework for building a windowing interface in a virtual environment. Proceedings of INTERACT: IFIP International Conference on Human-Computer Interaction, Washington, DC, USA, pp. 809–812.

[4] Dykstra, P. 1994. X11 in virtual environments: combining computer interaction methodologies. X Resource (Jan. 1994), 195-204.

[5] John Kelso et al, 2002. Diverse: A framework for building extensible and reconfigurable device independent virtual environments. Proceedings of the IEEE Virtual Reality Conference 2002, Washington, DC, USA, pp. 183.

[6] OpenSceneGraph. http://openscenegraph.org.

[7] Raimund Dachselt and Anett Hubner, 2007. Virtual Environments Three-dimensional menus: A survey and taxonomy. Computers & Graphics 31 (2007) pp. 53–65