VirtualExplorer: A Plugin-Based Virtual Reality Framework

Falko Kuester, Bernd Hamann, and Kenneth I. Joy

Center for Image Processing and Integrated Computing
Department of Computer Science
University of California, Davis, CA 95616-8562
{kuester, hamann, joy}@cs.ucdavis.edu

ABSTRACT

This paper introduces VirtualExplorer, a customizable plugin-based virtual reality framework for immersive scientific data visualization, exploration and geometric modeling. The framework is layered on top of a run-time plugin system and re-configurable virtual user interface and provides a variety of plugin components. The system provides access to scene-graph-based APIs, including Performer and OpenInventor, direct OpenGL support for visualization of time-critical data as well as collision and generic device managers. Plugins can be loaded, disabled, enabled or unloaded at any time, triggered either through pre-defined events or through an external Python-based interface. The virtual user interface uses pre-defined geometric primitives that can be customized to meet application-specific needs. The entire widget set can be reconfigured dynamically on a per-widget basis or as a whole through a style manager. The system is being developed with a variety of application areas in mind, but its main emphasis is on user-guided data exploration and high-precision engineering design.

Keywords: Virtual Reality, Immersive Environments, Virtual Reality APIs, Scientific Data Visualization and Exploration.

1. INTRODUCTION

Virtual reality systems have been around for almost a decade and have demonstrated the advantages and disadvantages of a variety of different design philosophies. A variety of virtual reality APIs have been developed in academia during the last couple of years including systems such as Avango, CAVE Lib, Lightning, MR Toolkit, ViewSVR and VrJuggler [2,3,4,10,16]. VirtualExplorer has its early roots as far back as 1993 and has evolved over different generations and transitioned through different evolutionary stages. The system is being developed by the Virtual Environments Group of the Center for Image Processing and Integrated Computing at the University of California at Davis and was designed as a flexible and intuitive scientific visualization, exploration and geometric modeling system. The VirtualExplorer framework aims at providing an easily understandable, yet feature rich development interface to the novice VR user. It has entered the pre open-source stage and will be available to the public after final testing is completed.

The motivation for creating such a system however, has its roots much deeper than just the creation of three-dimensional (3D) visuals from pre-computed data or interactive modeling. During most research and development projects the visualization component goes far beyond displaying final results. Visual results are used throughout the early test cycle to explore and verify simulation results or analyze and modify algorithms under development. In this respect, the almost unlimited three-dimensional workspace, offered by a virtual environment in combination with the right interaction metaphors, provides access to another flavor of intuitive debugging and development tools. Furthermore, academic research environments frequently face the "graduation-killed-the-code problem" in which significant amounts of invested research time are instantly turned into unmaintainable legacy code. Our system serves as an aid in establishing a common development base, which enforces coding standards, revision control and collaboration-oriented research avoiding staggering amounts of redundant work. Therefore, we need a very intuitive and easy to integrate framework allowing graphic developers to write vr-enabled code. On the other hand, the system must be flexible enough to provide access to a wide variety of physical and simulated input and output devices. Furthermore, it must function well in both its vr-mode and the traditional keyboard and mouse based on-screen mode. Performance, performance testing and debugging in general must be possible in both of these modes. Another important concern was that the system should be extensible without in-depth knowledge of the underlying framework. Since the system is intended for the visualization of large-scale scientific data, failure tolerances play

a crucial role, making it important that individual components in the application can be restarted, reconfigured, added or removed during execution.

2. SYSTEM SPECIFICATIONS

This section introduces VirtualExplorer, a highly re-configurable plugin-based VR framework for intuitive two-handed geometric modeling and data exploration. The open-source framework provides a variety of standard VR components such as a scene-graph-based API for static objects, an OpenGL rendering engine for the visualization of time-varying data, a re-configurable virtual user interface and a run-time plugin system. System components are depicted in Figure 3. The core design provides technical and non-technical users with an easy-to-use framework for the creation of realistic and content-rich environments and the tools for the exploration of scientific data sets. At the same time, it was important to offer an unconstrained *at-scale* interface to the user that reduces or removes the pre-meditative design phase. This can be accomplished by providing an environment that fosters the use of verification tasks and the development of modeling strategies as part of the design cycle, resulting in a thoroughly developed and tested final product. Visibility, reachability and accessibility controls must be built-in features in such an environment and should be automatically used throughout the design cycle. Furthermore, the system must provide the means for tracking the model creation history in combination with user-specifiable viewpoints or design paths. VirtualExplorer is being developed with a variety of application areas in mind, but its main emphasis is on user-guided data exploration and high-precision engineering design. From these initial design considerations the following list of requirements evolved:

- Ease of use
- Flexibility
- Performance
- Precision
- Extensibility
- Scalability
- Maintainability
- Failure tolerance

The resulting framework is build on top of a run-time plugin system that manages the individual modular system components, a hardware abstraction layer, which provides a higher-level, hardware-independent interface and a re-configurable virtual user interface providing access to control features. The system also provides access to scene-graph-based APIs, including Performer and OpenInventor and direct OpenGL support for visualization of time-critical data as well as collision and generic device managers (Figure 1).

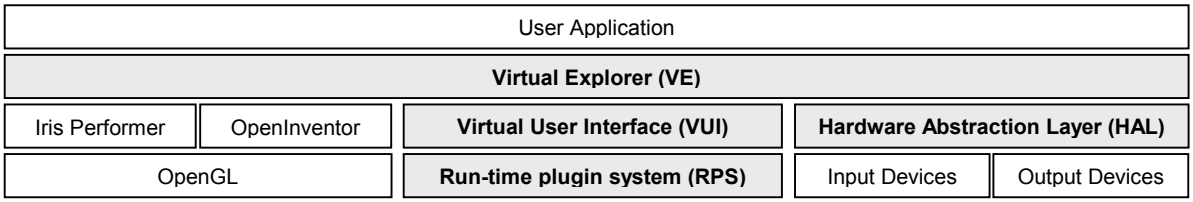

Figure 1: Application layers.

1. Run-Time Plugin System (RPS)

The RPS serves as the central building block for the VirtualExplorer plugin harness, which allows the user to control the behavior of both the system and its individual modules at run time. During the application startup phase VirtualExplorer uses RPS to load a set of standard system, device, utility and user application plugins located in predefined resource directories (Figure 2). When initializing system plugins, the harness verifies the correct operation of all crucial functions and, if possible, resorts to available fallback resources when core plugins fail to load successfully. The status of the plugin harness can be observed either through a registered plugin harness listener or queried and displayed with the help of the Python-based command line interface. The plugin system was designed to provide a high degree of flexibility during program execution, allowing plugins to be loaded, disabled/enabled, replaced or unloaded at any time, triggered either through predefined events or the command-line interface.

We distinguish among different flavors of plugins in order to maintain a clean separation between power users and novice application programmers. The plugin harness library provides the basic tools for the creation and control of system, device, utility or user application plugins. Each plugin is derived from an abstract plugin class, overloads a distinct set of functions, and provides a set of standard query functions used to determine its name, version, author, purpose and compatibility. Under

Linux and Unix, plugins are implemented as dynamically shared objects (DSOs) and dynamic link libraries (DLLs) are used for the different Microsoft Windows flavors. In general, user programs are written as application plugins that gain access to the vr-enabled framework at minimal cost. Simple applications can be created in minutes by deriving from one of the provided application plugin classes. Most commonly, users will implement rendering or frame plugins that are either adding visual context to the scene or are performing some sort of computation outside the drawing process. The plugin concept fosters the creation of modular and, most importantly, reusable components that can help in building a dynamic, feature-rich framework.

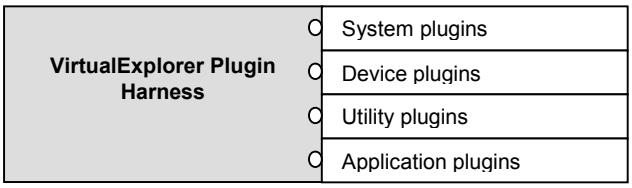

Figure 2: Plugin categories.

2. Virtual User Interface (VUI)

Menus are a vital component of all modeling systems since they provide access to the available system functions. With the transition from a 2D to a 3D environment, a new set of VR input devices and, consequently, new concepts must be implemented. Different solutions to this problem opt for either a direct port from the classical 2D menu to its 3D counterpart or new implementations designed specifically for 3D [6,14]. The VUI is composed from pre-defined geometric primitives/models that can be altered or replaced by the user, behavioral descriptions and specific component based actions. The user interface can be customized to meet application-specific needs or user preferences. A style manger is available that controls the overall look and feel of the complete VUI or individual components by supporting a variety of different base models available for each widget. The entire widget set can be reconfigured dynamically on either a per-widget basis or as a whole through the style manager. In combination with the layout manager, which controls widget placement, the creation of complex and appealing menus is possible. All container classes, widgets and layout managers are editable and can be individually controlled through a designated menu manager. The menu manager maintains the context information for all widgets currently part of the visual user interface and user plugins can attach to it to register their own specific visual representation and control characteristics. While the default VUI supports the most important system operations, more advanced components can be added through customized plugins. Widgets can also be associated with different visualization metaphors and linked to the user's hands, head and body or statically attached to the display using a heads-up-display (HUD) metaphor.

3. Plugin Managers

By design the VirtualExplorer kernel is kept as small as possible. It is responsible for handling the main control loop, process synchronization and system integrity. The run-time plugin manager is the core component of the VirtualExplorer architecture. The plugin manager provides the plugin control loop and is responsible for handling all operations associated with plugins. Through this manager, systems or user plugins can register a set of event listeners, which subsequently are called at the appropriate times. A Phyton-based interface provides control over operations such a plugin load, remove, replace, suspend, and resume at run-time. The bulk of the system functionality is added through system plugins that manage most of the relevant components. The base set of system plugins include the

- Device manager: manages all devices
- Input manager: manages all input devices
- Output manager: manages all output devices
- Menu manager: manages user interface components and interactions
- Render manager: manages drawing and swapping-related issues
- Display manager: manages system-specific display devices
- Collision manager: manages collision events between components in a scene
- Network manager: manages network plugins

The input, output and display managers serve as the standard interface point for arbitrary physical and simulated devices and they are implemented on top of the hardware abstraction layer. All devices in VirtualExplorer belong to either one of these three categories, which in turn are managed by a central device manager. The input manager handles multi-sensory input from physical or simulated devices such as the mouse and keyboard for monitor-based vr, as well as dedicated vr devices including spatial trackers and gloves. The output manager generally controls the process flow for multi-modal interaction feedback through haptic and acoustic devices. In addition, the display manager is responsible for creating the appropriate output channels for the used device and configuring the display properties.
User interfaces in VEs have to be highly configurable since look, feel and functionality significantly change with each application. Therefore, it is important to determine and provide access to the most relevant task-dependent functions. The menu manger handles all user-interface-specific events. VUIs can be created using the VUI framework described earlier and controlled through the menu manager. The render manager provides the required support for the available graphics APIs and handles the draw-loop-specific operations such as updating transformation matrices and clearing and swapping the frame buffer. The collision manager tracks collision events between different objects and also supports an object-specific "snapping" mechanism. An object is automatically attached to another object if it is within a certain distance to the other object and both objects have the same snapping parameters. The network manager serves as a central connection point for separately executed local or remote plugins and provides the required support for collaborative virtual environments. Furthermore, a base set of device plugins supporting the most common tracking and input devices such, as the Polhemus Fastrak, the Ascension Flock of Birds, the Fakespace PinchGlove and haptic devices such as the Phantom from Sensable are provided. New devices can be added easily through device plugin templates that allow device integration without in-depth knowledge of the underlying system.

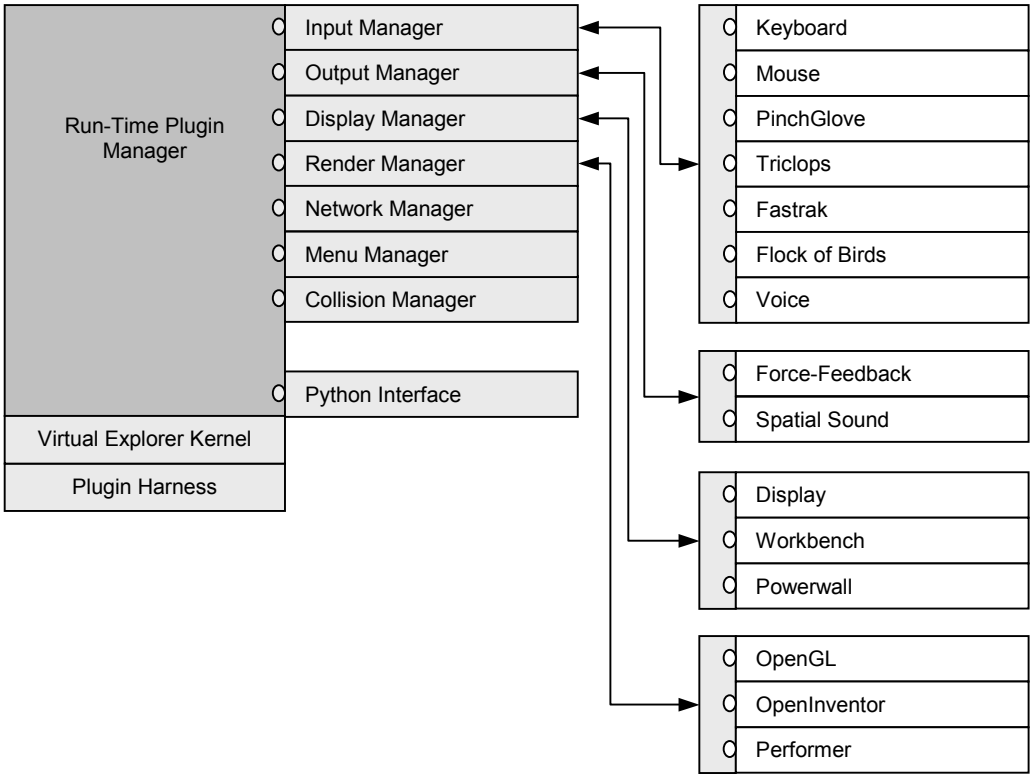

Figure 3: VirtualExplorer application framework.

4. Inter-Plugin Communication

The system uses an "event-emitter" and "event-listener" approach. Any application component can be turned into an emitter or listener that sends or receives particular events. Event listeners can be registered with an event emitter and subsequently informed whenever the particular event occurs. In general, event listeners are defined as abstract classes containing a set of

virtual methods that can be called by the event emitter. Application code can instantiate these listener classes to gain access to the desired callbacks and to register itself with the appropriate event emitters (Figure 4).
An extensive set of emitter and listener classes for the development of new system, device and utility plugins is provided and can be used for the accelerated integration of new components. In addition, plugin components can acquire handles to other plugins in the system and directly access their public methods. This approach is of benefit for the integration of smaller modular toolkits that do not require a transparent interface mechanism or should be shielded from the outside.

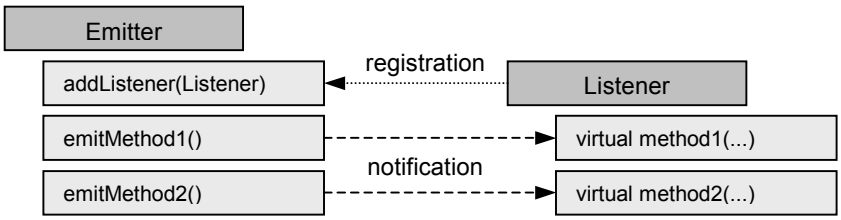

Figure 4: Emitter-listener-based event handling.

5. Hardware Abstraction Layer (HAL)

VirtualExplorer strictly distinguishes between system-specific device drivers and device interfaces through its well defined hardware abstraction layer. Its purpose is to provide a platform- and device-independent API to the user that provides access to all relevant device-related data without having to deal with implementation-specific issues. The application developer only deals with the specific high-level interfaces to devices providing, for example, positional, orientation, and state or event information. Device drivers in turn must implement one or more of the predefined interfaces and follow their particular specifications. All device drivers are implemented as stand-alone processes that can send or receive data by utilizing VirtualExplorer's shared memory or network interfaces. Each device has to identify its unique vendor string and specify which of VirtualExplorer's device types it supports. The specified device type or types precisely determine which data it can send or receive. This mechanism provides developers with the means to add new device drivers with minimum effort. It also caters to the hardware vendors that, for commercial reasons, might not be able to follow the open-source concept, by allowing them to provide device plugins in binary form. A user can gain access to individual devices by simply instantiating the appropriate interface and specifying the device vendor string. A handle to a six-degree-of-freedom tracking device, for example, can be created through an instance of the VeTracker class, which then is used to register a tracker-specific listener with the input device manager. Subsequently, whenever the device provides updated data the appropriate method in the listener is called (Figure 5). Following the idea of the HAL, the user does not actually create a handle to that particular hardware device but a handle to a generic tracking device listed under the specified name. The device delivering the data, however, can be the actual hardware device specified, one that creates similar data, or a simulator implemented in software.

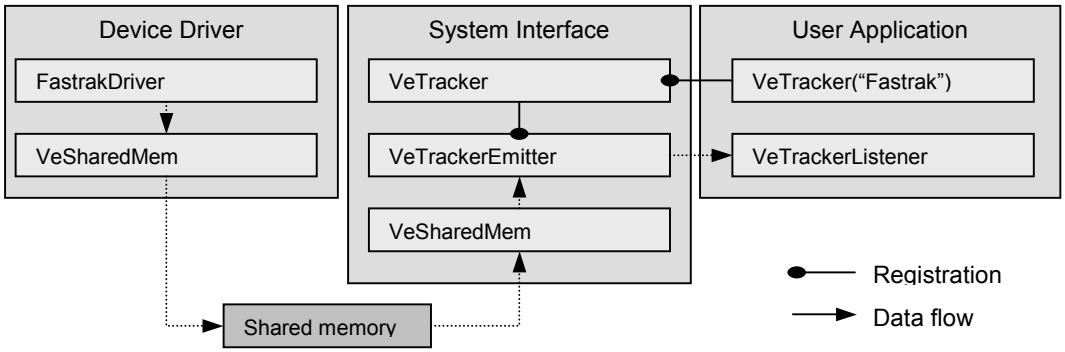

Figure 5: Example of device abstraction layer and message passing.

6. Visual Debugging Support for Virtual Environments

One of the problems frequently encountered during the development of vr systems is that of testing, debugging and performance tuning all potentially complex processes of the system. The modular plugin-based approach allows the user to simplify the studied systems by enabling and disabling components during execution time. This approach can greatly help in identifying potential problems and the analysis of bottlenecks. However, this feature also caters to the developers of new algorithms by allowing them to visualize their results and to modify application code on the fly, without having to restart the system. This concept opens the door for the development of entirely new suites of plugin-based interactive development tools.

3. CONCLUSIONS

VirtualExplorer provides an open source framework for the accelerated development and testing of complex virtual environments. It currently offers the user a vr-enabled and Python-scriptable run-time plugin system (RPS), a hardware abstraction layer (HAL), drivers for a variety of tracking devices, a virtual user interface API (VUI) and support for rendering APIs, including OpenGL, Iris Performer and OpenInventor. The system was designed to be easily extensible through additional plugin components.

4. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of ALSTOM Schilling Robotics and SGI, and thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

5. REFERENCES

1. Agrawala, M., Beers, A. C., Froehlich, B., Hanrahan, P., McDowall, I. And Bolas, M., "The two-user responsive workbench: Support for collaboration through independent views of a shared space", in SIGGRAPH 97 Conference Proceedings, T. Whitted, ed., Annual Conference Series, pp. 327-332, ACM SIGGRAPH, Addison Wesley, Aug. 1997.
2. Bierbaum, A., "VrJuggler: A Virtual Platform for Virtual Reality Application Development." MS Thesis, Iowa State University, 2000.
3. Conway, M., Pausch, R., Gossweiler, R. and Burnette, T., “Alice: A Rapid Prototyping System for Building Virtual Environments,” Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems, Short Papers: Designing Interaction Objects, Vol. 2, pp. 295-296, 1994.
4. Cruz-Neira, C., "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, May 1995
5. Cutler, L. D., Froehlich, B., Hanrahan, P., "Two-handed direct manipulation on the responsive workbench", in Proceedings of the Symposium on Interactive 3D Graphics, pp. 107-114, ACM Press, (New York), Apr. 27-30 1997.
6. Deering, M. F., "The HoloSketch VR sketching system", Communications of the ACM, 39(5):54-56, 1996.
7. Durlach I., Mavor, A.S., Committee on Virtual Reality Research, Commission on Behavioral Development, Social Science, Mathematics Education, Commission on Physical Sciences, and Applications, National Research Council. “Virtual Reality: Scientific and Technological Challenges,” National Academy Press, 1994.
8. Ebert, D. S., Shaw, C. D., Zwa, A. and Starr, C., “Two-handed interactive stereoscopic visualization”, in: Proceedings of IEEE, R.Yagel and G. M. Nielson, eds., pp. 205-210, IEEE, Los Alamitos, Oct. 27-Nov. 1 1996.
9. Galyean, T. A., “Guided navigation of virtual environments”, 1995 Symposium on Interactive 3D Graphics, pp. 103-104, April 1995.

10. Ghee, S., and Naughton-Green, J., “Programming Virtual Worlds", ACM SIGGRAPH '94 Course, 17, 1994
11. Green, M. and Halliday, S. , A geometric modeling and animation system for virtual reality, Communications of the ACM 39, pp. 46-53, May 1996.
12. Green, M., "Shared virtual environments: The implications for tool builders", Computers and Graphics 20, pp. 185-189, Mar.-Apr. 1996.
13. Guiard, Y. and Ferrand, T., "Asymmetry in bimanual skills, in: Manual asymmetries in motor performance", D. Elliott and E. A. Roy, eds., CRC Press, Boca Raton, FL., 1995.
14. Haeffner, U., Simon, A. and Doulis, M., “Unencumbered Interaction in Display Environments with Extended Working Volume”, In Stereoscopic Displays and Virtual Reality Systems VII, John O. Merrit, Stephen A. Benton, Andrew J. Woods, Mark T. Bolas, Editors, Proceedings of SPIE Vol. 3957, pp. 473-480, 2000.
15. Krueger, W., Froehlich , B., “Visualization blackboard: The responsive workbench”, IEEE Computer Graphics and Applications 14(3):12-15, May 1994.
16. Landauer, J., Blach, R., Bues, M., Roesch, A. and Simon, A., "Toward Next Generation Virtual Reality Systems", Proc. Of the IEEE International Conference on Multimedia Computing and Systems, 1997
17. Shaw, C. and Green, M., "The MR Toolkit Peers Package and Experiment.." IEEE Virtual Reality Annual International Symposium (VRAIS 93), pp 463-469, 1993.
18. Watsen, K., and Zyda, M., “Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments”, IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, Georgia, 1998.