Eurographics Symposium on Virtual Environments (2006)
Roger Hubbold and Ming Lin (Editors)

vjVTK: a toolkit for interactive visualization in Virtual Reality

K. J. Blom[1] [1]interactive media/virtual environments, University of Hamburg, Germany

Abstract

vjVTK is a small toolkit enabling the use of the Visualization ToolKit (VTK) natively within the VRJuggler Virtual Reality framework. The toolkit enables a departure from the traditional visualization calculation-conversion-immersive viewing cycle. vjVTK leverages the OpenGL capabilities of VTK to allow it to run as a native graphics generation tool within the VRJuggler framework. This removes the need for offline visualization generation and format conversion. Most importantly, vjVTK introduces the ability to interactively change the visualization, exploiting VTK's full capabilities and providing researchers with a more powerful tool.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism I.3.8 [Computer Graphics]: Applications

1. Introduction

Scientific Visualization has become a mainstay application for Virtual Environments. The visualization of data and phenomena enables scientists to understand their data and results by exploiting the power of the human visual system. While exploring scientific data in Virtual Reality (VR) settings is becoming a more commonly used tool for scientists from various fields, the programming requirements involved in creating these environments forms a large barrier for scientists. The most frequent method for scientific visualization in VR is a visualization generation - format conversion - immersive viewing cycle, where static visualizations are created in the visualization software, converted to a format for the VR system, and finally viewed. When parameters need to be adjusted, as is often the case, the cycle must be repeated, since the adjustments are made in the visualization software. In this paper, we present a software library, vjVTK, designed to further simplify this process for scientists, while also enabling true interaction with visualizations.

2. Background

Scientific Visualization has become a well developed field. Various existing commercial applications and open-source programs, such as the Visualization ToolKit [SML98], allow scientists to visualize their data in meaningful ways. This transformation of the data enables the scientists to exploit the power of the their visual systems, thereby assisting their understanding and analysis of their data, an example can be seen in Figure 1. In extreme cases, the visualizations allows scientists to discover new information previously hidden in the massive amounts of data [Tufo99]. Visualization programs commonly provide desktop visualization support, but often no or only limited support for Virtual Reality (VR) displays. In order for the scientists to exploit the power of immersive VR display systems, the visualization software must be somehow coupled with a VR system.

A few methods are available for viewing scientific visualization in VR displays. The most predominate is processing the data within the scientific visualization program, producing 3D geometric information, which is in turn displayed in a VR system. Commercial products, which often are focused on specific research areas, commonly provide the ability to export to various standard 3D formats. For the general visualization tool, the Visualization ToolKit (VTK), a popular converter exists. It converts VTK's visual entities, Actors, to a SGI Performer™ scene-graph format, vtkActor2Performer `http://brighton.ncsa.uiuc.edu/~prajlich/vtkActorToPF/`. In all cases, these converters enable immersive exploration of the generated static visualizations. Moreover, this conversion method generally is off-line and, therefore, does not allow for user interaction with the visualization process. In vtkActor2Performer's case, with some effort it can be embedded into the VR system render loop, creating interactivity at the cost of the conversion process.

While in some applications immersively navigating the information space may yield results, in many applications interactively changing values in the visualization process would be more instructive. A simple example of this, occurring often in scientific visualization, would be moving a cutting plane through a field. With the conversion method, this becomes possible only through generating the cutting plane for every position and then picking the correct texture at run-time. A further detractor of this method is that the conversion method often adds the overhead of learning how to use a scene graph system, required only for the display of the visualization. While for many in the VR community this is a light endeavor, for many scientists and researchers this is a superfluous and difficult proposition.

The other possbile method is building the visualization software directly into the VR system. This method has many natural benefits. Having the visualization generate geometry directly in the VR system reduces development cycles and, dependant on the visualization software, introduces the possibility of interactivity with the visualization generation. The largest notable drawback is that VR is highly dependant on the graphics generation being performed in real-time. Unfortunately, many of the visualization techniques require longer times to be calculated. This problem is exasperated by the introduction of interaction with the visualization. When the interaction requires updating the visualization, the frame rate of the VR system can be significantly undermined. This method of combining VR and visualization has been used for various commercial products and in some locals for their individual use. In most cases the VR component is limited, particularly in its flexibility. One notable commercial product for general visualization, Amira [Mec], fits into this category. As it is built on a scene-graph system, it usable in VR through an additional plug-in.

3. Design and Implementation

The design goals of vjVTK focus on making open-source based visualization in VR easier, particularly for engineers and scientists. To achieve this, vjVTK embeds VTK into an open-source VR software framework, VRJuggler [BJH*01]. The design is based on the premise of the undistributed vtkCAVE [TFPB99], a library embedding VTK into the cavelib libraries developed at Argonne National Laboratories. The marriage of VTK's native OpenGL based visualization system with VRJuggler's hardware and display system abstractions creates a powerful and flexible system. Users have only to learn VTK and a minimal part of VRJuggler to take advantage of visualization in VR and gain the ability to add interactivity.

The biggest challenge in combining the two such systems is that both expect to have control of the rendering pipeline, specifically the windows and contexts. VTK expects not only to open the windows and control viewing parameters, it queries and uses the current context often in its rendering and interaction processes. VRJuggler, to provide its abstraction layers and flexibility, must control the windowing, contexts, frustum, etc.

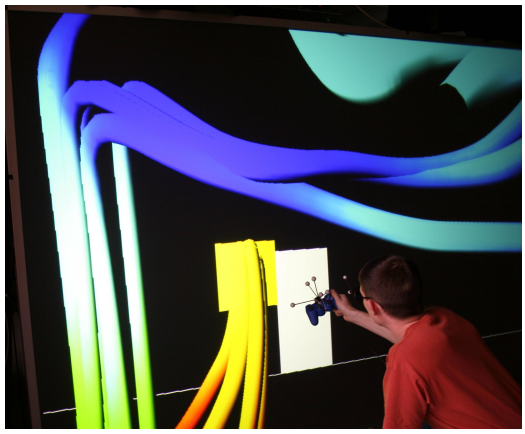

Figure 1: *A user exploring streamtubes representing the airflow through an office space in an adaptation of one of VTK's standard examples. The source of the streamtubes is selectable by the user and can be moved, altering the seeding and creating new flow paths through the environment.*

The core of vjVTK is the classes which handle this meshing of the systems, consisting of: VTK derived RenderWindow and Renderer classes, a Renderer facade, and VRJuggler derived GlApp classes. The VTK derived classes override virtual functions, which deal with frustums and context sensitive code. A few new functions allow the VTKApp, a class derived from GLApp, to set some parameters in a non-traditional way. The VTKApp classes then setup and call the Renderer and RenderWindows at appropriate places in the frame loop, invisible to the user. This solution works, but one difficulty remains. The user must add or delete Actors to each Renderer, an OpenGL context sensitive function call. In VRJuggler, the context is only valid during the draw callback, creating a complex and messy coding environment. Unfortunately, this dependency is buried in private non-virtual functions of the underlying VTK classes. To circumvent this, a facade design pattern was introduced for the Renderer class, vjRenderer. The user simply add/removes their Actors to this Renderer. The facade operates in conjunction with the VTKApp classes to make all changes at the appropriate times in the frame-loop.

vjVTK also provides VTK based Pickers and Interactors, which respectively enable ray picking and the use of special interaction classes to simplify interaction programming. Use of the Pickers to select Actors or individual cells is straight forward and interaction with the VTK pipeline can be coded by the user in the usual ways. To simplify this for the end user, VTK has special interaction sets, combining Interactors and Widgets, which provide prototypical in-

teraction metaphors. Such a development is naturally usefull for a user friendly design. Currently in development for vjVTK is a series of example Widgets, e.g. the PlaneWidget which allows a plane to be translated, scaled, rotated, where the plane can be a source within the visualization pipeline (see Figure 1). Unfortunately, the interaction methods implemented in the Widgets are strong dependant on VTK's desktop environment. In converting each Widget implementation to VR, they become highly dependent on the built-in interaction metaphor; more troubling is that writing the Widgets is not straightforward. The examples are provided to give a starting point for advanced users to develop their own Widgets. Functional examples at the time of writing include the PlaneWidget and BoxWidget.

One of the benefits of working with converters has been, ironically, working in the scene-graph. Visualizations systems, such as VTK, typically focus on the scientific visualization problem and only that problem. Visualizations often have components, which are not part of the simulation, particularly in VR the context surrounding can be important. VTK does include some ability to place additional geometry in the environment; however, it is not truly designed for that nor for interaction with such objects. Working with a scene-graph for additional graphics made this easier, and this advantage is lost when switching to a purely VTK based rendering. vjVTK offers a way to assist in this case. Since VTK and scene-graphs, such as OSG, `http://www.openscenegraph.org`, and OpenSG, `http://www.opensg.org`, are OpenGL based, they can be combined in the same rendering loop. We have created two application classes, combining VTK with the forementioned scene-graphs. At this point the two systems are unaware of each other, not even sharing lighting. As OpenGL is state machine, this means that each system must reset the correct state each frame. These inherent limitations will be addressed in a coming version of vjVTK.

As the impetus driving vjVTK's development is making visualization in VR more accessible, we will here discuss briefly how the user sees vjVTK. With vjVTK, the rendering aspect of VTK becomes more transparent to the user than in traditional VTK usage. VRJuggler handles all of the setup of windows and input devices, freeing VTK users from this duty. As is usual in working with VRJuggler, two main functions are required to be filled in by the users, initScene and preFrame. The initScene is the function call where the user establishes their VTK pipeline, ending with the addition of Actors to the special vjRenderer provided by the base VTKApp. The preFrame function is called before each render frame, allowing calculations and changing of the VTK pipeline, e.g. changing the position of a cutting plane. One difficulty which remains on the shoulders of the user is maintaining a update rate that is reasonable for VR. vjVTK removes some of the overhead from other methods, but visualization calculations remain time consuming. Performing changes to the VTK pipeline in the form of interaction may cause serious frame-rate problems. Fortunately, VTK has built-in functionality to attempt to limit the affected portions of the pipeline, giving the user some relief.

3.1. Conclusions

In this paper we have introduced vjVTK, an embedding of the Visualization Toolkit into the VR framework VRJuggler. This enables researchers to easier and more quickly exploit the interactive power of VTK, while providing the ability to use VR technology. The VRJuggler interface provides abstraction from the display and flexibility in display technology which the VTK user previously lacked. vjVTK supports interactivity through VTK derived methods including Pickers and Interactor/Widget based interaction methods. Additionally, researchers benefit as they need only learn VTK and a small portion of VRJuggler.

vjVTK is still a work in progress. While the base functionality is present, there are improvements to be made. As mentioned in the discussion above, examples of interaction Widgets are under development currently. A method in which to make VTK and the deployed scene-graph aware of one and other is needed, at least in so far as lighting is properly done between the systems. As one of the goals of this software package is to make visualization in VR easier for users, particularly for non-computer scientists, we are exploring the possibility of introducing Python bindings for vjVTK. This could potentially be done by coupling and extending the pyJuggler extension with the Python language binding of VTK. Users could then program their pipelines in Python instead of C++, which for many, would be a lighter language to learn.

References

[BJH*01] BIERBAUM A., JUST C., HARTLING P., MEINERT K., BAKER A., CRUZ-NEIRA C.: VR Juggler: A Virtual Platform for Virtual Reality Application Development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)* (Washington, DC, USA, 2001), IEEE Computer Society, p. 89.

[Mec] MECURY COMPUTER SYSTEMS INC.: Amira. `http://www.tgs.com` checked Mar 31, 2006.

[SML98] SCHROEDER W., MARTIN K. M., LORENSEN W. E.: *The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[TFPB99] TUFO H. M., FISCHER P. F., PAPKA M. E., BLOM K. J.: Numerical simulation and immersive visualization of hairpin vortices. In *Proc. SuperComputing '99* (1999), IEEE Computer Society. (ANL archives) `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P779.ps.Z`.