Immersive Visualization using AVS/Express

Ian Curington

Advanced Visual Systems Ltd.
Hanworth Lane,
Chertsey, Surrey
KT16 9JX
UK

Phone: +44-1932-581600
Fax: +44-1932-581600
Email: ianc@avs.com
Website: www.avs.com

Keywords:
Visualization, application frameworks, immersive environments, computational steering.

Immersive Visualization using AVS/Express

Ian Curington
Advanced Visual Systems Ltd.
Hanworth Lane, Chertsey, Surrey, KT16 9JX UK
ianc@avs.com

Abstract. To achieve benefit and value from high performance computing (HPC), visualization is employed for a direct understanding of computational results. The more closely coupled the visualization system can be to the HPC data source, the better the chance of fostering discovery and insight. A software architecture is described, with discussion of the dynamic object manager allowing scalable high-performance applications and computational steering. High performance parallel graphics methods are described, including interactive HPC data exploration in immersive environments.

1. Introduction

Visualization systems that aim to address complex data visualization problems, large data sources, while at the same time providing interactive immersive presentation, must be scalable, highly configurable and provide mechanisms for efficient data access and high performance display. The AVS/Express visualization software system addresses these requirements by using a runtime dynamic object manager. Methods of managing large data access, computational steering, parallel multi-pipe rendering and immersive menu and interactive device control systems are implemented using the object manager framework.

2. High Performance Framework

The AVS/Express visualization system is based on a framework for rapid prototyping and deployment of complex applications. Although visualization is the primary content of the software object library, it has been used in a wide variety of other areas.

2.1 Scalable Light-Weight Object Manager

At the heart of the framework is the object manager (OM). This performs the roles of scheduling, object state control, object notification and event management. An object is an abstract class, as small as a single byte or string and as large as a complete application container. As most applications have a large number of objects, the OM is highly tuned for efficient management. A current application for the European Space Agency (ESA) has as many as 750,000 visible objects for example.

Object relationships are hierarchical, both in class inheritance structure and in instance application design structure. Objects are moveable between processes and hosts. Applications often use dynamic object control so that only those objects used by the current application state are instanced.

The OM and visualization module implementations allow for full 64-bit address space support, including the layers that pass data between processes. This allows the system to scale to very large data problems.

2.2 Dynamic Data File Cache Management

A set of "file objects" are included that help to manage data file access during visualization. These dynamically map parts of files to memory arrays. This way references to the arrays by the rest of the system trigger dynamic file access, allowing files larger than virtual memory to be processed. This system is similar to an application specific cache management layer. Data sizes of 200 Gbytes have been processed using the file object system.

3. Computational Steering

As much of the data used in high-performance visualization systems originates from simulation, there is a natural interest in coupled simulation-visualization systems. Using a modular visualization system, this can be achieved in a simple and easy way. By making minor changes to MPI parallel simulation programs, they can be steering enabled, whereby the visualization system can monitor simulation progress and provide interactive user control over simulation parameters [6]. New insights may quickly be gained by such continual monitoring and guiding the progress of computational simulations that were only previously analyzed in their final state.

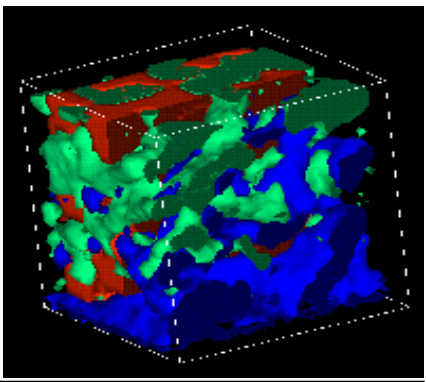

Figure 1 Result of a steered simulation to determine the spinodal point of an immiscible fluid.

Another system for computational steering, ViSIT [4] is a set of utility functions intended to support online-visualization and application steering. The current system has C and FORTRAN bindings for the application end (or 'simulation') and supports AVS on the visualization side. The idea is that the simulation is 'more important' than the visualization. Therefore the

simulation triggers all actions. The simulation actively establishes a connection and sends or receives data.

Given this structure, a ViSIT-enabled simulation can establish a connection to a visualization/steering application, transfer 1,2,3 or 4D-data to and from that application, then shut the connection down again. On the AVS visualization side, the interactive user can receive and send data from and to the simulation while allowing data that arrives from the simulation to trigger visualization modules.

4. Parallel Rendering

In AVS/Express, the very flexible graphics architecture is exploited to minimize memory requirements and data structure conversions during rendering for large data problems. A thin layer OpenGL renderer provides a graphics display path, without dependence on scene tree, dynamic scene creation, or any requirement to hold the scene content in virtual memory. For small data sets and high speed interaction, stored scene tree mechanisms are used, but not required.

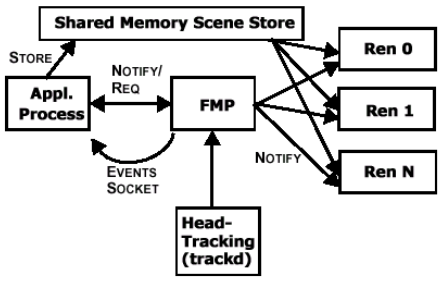

Figure 2 Frame Manager Process implements both pipeline and multi-pipe rendering parallelism.

Data source methods, and visualization methods are able to register user-defined draw methods providing a "chunking" mechanism to build up the scene using a multi-pass or data segmentation model. Visualization procedures are only executed as needed during the incremental updates, and do not need access to the entire model. In addition, this architecture allows procedural rather than stored data structure sources. By coupling this system to file access APIs, very large data sources may be managed in an efficient manner.

4.1 Multi-Pipe SDK

The SGI Multi-Pipe SDK is designed as a new problem-solving API [1]. The design is to allow OpenGL applications to easily migrate to multi-pipe computing and graphics environments. The system supports full portability, so the same application can run on a single display low-end desktop system, through to an Onyx2 with multiple Rendering Engines. Unlike Inventor or Performer, the Multi-Pipe SDK does not impose a scene-graph structure, so dynamic update of scene contents, as often happens in visualization applications, is handled in an efficient way. The system provides a transparent, call-back driven programming interface, and takes care of inter-process

communication, parallel execution, and display configuration issues. The system is used to implement full parallel rendering within AVS/Express Multi-Pipe Edition [2].

The AVS/Express system is a full-featured visualization system and application development environment. In addition to over 850 visualization operators and interfaces, it has a generalized viewer and interaction system for the display of visualization data. The viewer is split into device (graphics API) independent and dependent layers. The mapping between these layers uses a "virtual renderer" dynamic binding structure. In this way, multiple renderers can be supported in the same runtime environment. The AVS/Express Multi-Pipe Edition extends the virtual renderer base class, and adds callbacks to the Multi-Pipe SDK subsystem.

4.2 Multi-Pipe / Multi-Channel Display

To achieve flexible support for many virtual display environments, full multi-pipe and/or multi-channel display control is mapped onto the parallel rendering process. This allows display on a wide class of VR systems, such as Immersadesk™, HoloBench™, CAVE™ or panoramic multi-channel displays.

4.3 Pipeline Parallelism

To implement full parallel rendering, an intermediate Frame Management Process (FMP) was introduced to separate the handling of the rendering from the application process [5]. When a frame is ready (fully tokenized) the application notifies the FMP which in turn notifies the MPU processes and is, itself, blocked. The application continues with its tasks and produces the next frame. Note that the application process is eventually blocked, so that there is no build-up of frames should the production of frames be faster than the consumption rate of the renderers. Meanwhile all rendering processes run in parallel for simultaneous update of all display channels.

4.4 VR Menu System

A menu system for virtual reality and immersive stereo applications is included that allows for rapid prototyping, configuration changes, and 3D-layout behavior management. The VRMenu user-interface toolkit exploits 3D geometric graphics systems and allows interactive immersive data exploration without leaving the environment to modify control parameters.

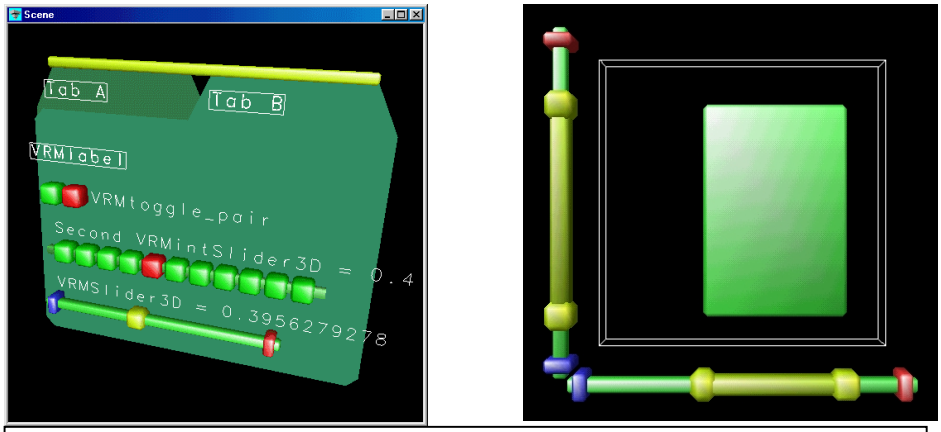

Figure 3 3D Menu manipulation tools – tab panels, buttons, choice lists, sliders and region crop controls.

A primary goal of interactive data visualization in an immersive environment is to get inside the data space for exploration and discovery. Ideally, no other distractions should be introduced, allowing full attention to be directed to the detail in the visualization. The VRMenu system is used to replace the 2D GUI with a 3D geometric equivalent, so that key parameters in the visualization can be controlled within the immersive environment without interruption.

4.5 Trackers / Controllers

An important part of the immersive experience is tracking the 3D viewer position, correcting the perspective stereo views for optimal effect. The head-tracking process (using *Trackd* software from VRCO Inc.) passes viewer location information to the FMP [7]. For head-tracking support use of the FMP means that the camera can now be decoupled from the application, the FMP piloting the center of the view with camera offsets without interruption of the application. If a new frame is ready then the FMP will immediately use it, otherwise it will re-render the previous frame according to the new camera position. Figure 2 shows that the head-tracking data feeds directly into the FMP, which will trigger the renderers to redraw with the updated camera, without interrupting the main AVS/Express Multi-Pipe Edition application process.

5. Visualization Methods

A recently developed method of using a moving texture alpha-mask to represent scientific data is used for the purpose of visualizing continuous fluid dynamics fields [3]. The method combines stream tubes and particle animation into one hybrid technique, and employs texture surface display to represent time, flow velocity and 3D

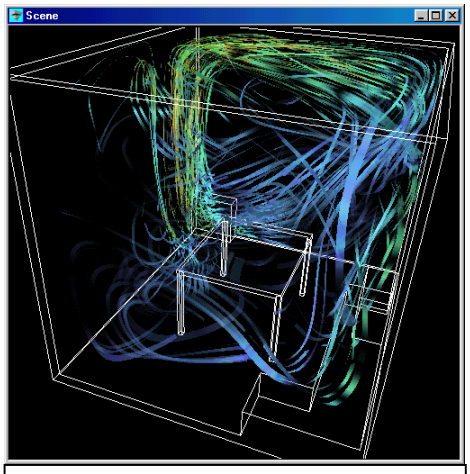

Figure 4 Convection Heat Flow in Room using Texture Wave Ribbons

flow structure in one view. The technique exploits texture support in high-performance graphics systems and achieves high graphics efficiency during animation.

This approach combines the particle tracing technique with the geometric stream ribbon and tube technique including the advantages of both. The spatial information is shown using geometry, while velocity variations over time are shown using an animated texture with variable transparency. The color used in the texture map is a direct replacement to scalar color assignment, yielding high quality color contours and avoids artifacts introduced by RGB interpolation during graphics display

One approach is to apply an offset to the *u-v* texture coordinates, creating an animation to make the texture crawl along the stream path. Here however, the u-v texture coordinates remain static, so the color and flow variables remain referenced to each other for analytical flow visualization, while the content of the texture alpha-mask is dynamically changed to produce an animation. In this way a continuous periodic function can be applied, rather than a direct positional shift. The center of highest opacity in the alpha-mask is used to create a time-wave pulse function. As phase is adjusted, the positions of the pulse function peaks move along the time path.

6. Conclusion

The AVS/Express visualization system has been described, highlighting the characteristics that address large data visualization. Through efficient data access and high performance parallel display, the underlying framework scales to handle large problems. Application architectures have been discussed for directly coupled high-performance computing applications with visualization for monitoring and steering of those applications through visual interfaces. Finally, the use of immersive visualization techniques has been shown to add value in the exploration of large data problems.

7. References

1. Bouchaud, P. “Writing Multipipe Applications with the MPU”, SGI EMEA Developer Program document, 1998. http://www.sgi.com/software/multipipe/sdk/
2. Curington, I. "Multi-Pipe Rendering Framework for Visualization Applications" Proceedings of SIGRAD ’99, Stockholm December 1999
3. Curington, I. "Animated Texture Alpha-Masks for Flow Visualization" Proceedings of IEEE IV2000, Information Visualization conference, London July 2000
4. W. Frings, T. Eickermann, “visit - VISualization Interface Toolkit”, Zentralinstitut fuer Angewandte Mathematik, Forschungszentrum Juelich GmbH http://www.kfa-juelich.de
5. Lever, Leaver, Curington, Perrin, Dodd, John, Hewitt, “Design Issues in the AVS/Express Multi-Pipe Edition. IEEE Computer Society Technical Committee on Computer Graphics, Salt Lake City, October 2000.
6. Martin, Love “A Simple Steering Interface for MPI Codes” *(to appear)*
7. VRCO Inc., Trackd tracker/controller device drivers http://www.vrco.com