

An Application Architecture for Large Data Visualization: A Case Study

C. Charles Law, Amy Henderson
Kitware Incorporated

James Ahrens
Los Alamos National Laboratory

Abstract

In this case study we present an open-source visualization application with a data-parallel novel application architecture. The architecture is unique because it uses the Tcl scripting language to synchronize the user interface with the VTK parallel visualization pipeline and parallel-rendering module. The resulting application shows scalable performance, and is easily extendable because of its simple modular architecture. We demonstrate the application with a 9.8 gigabyte structured-grid ocean model.

1 Introduction

The ability of massively-parallel simulations to generate data has outpaced our ability to visualize large data sets. The size of models generated by distributed simulations makes it difficult to collect the results on a single computer for visualization. To accommodate, simulations save less data for fewer time steps, and visualizations are generated less often. This solution has many unfortunate consequences. Key results can be missed or simulations gone awry can run for days before the problem is noticed. A simple, effective method for visualizing large data sets is needed to analyze the results of today's simulations.

Climate simulations by the Parallel Ocean Program (POP) at the Los Alamos National Laboratory (LANL) are a good example of the large-data problem. Current simulations are using a $1/10^{\text{th}}$ degree global grid. The resulting structured data set has dimensions $3600 \times 2400 \times 40$. The constant grid points require ~ 4.1 GB when reconstructed, and each 3D attribute requires ~ 1.4 GB. 10 year simulations with daily time step dumps are generating an overwhelming amount of data. Current visualization systems are unable to handle data sets of this magnitude.

0-7803-7223-9/01/\$10.00 Copyright 2001 IEEE

This case study discusses recent multiprocessing features added to the Visualization ToolKit and how they were used to create a turnkey application called ParaView. We demonstrate the application running on a 9.8 gigabyte ocean model.

2 The Visualization Toolkit

The Visualization ToolKit¹ (VTK) is an open source freely available software toolkit for 3D computer graphics, image processing and visualization^[14]. VTK has recently been extended to process visualization tasks in parallel^[2,8]. The demand driven pipeline can now operate on pieces of data sets: unstructured as well as structured. This allows out-of-core streaming of large data over time, and distributing data sets over multiple processes for data parallelism.

In VTK, communication between processes uses a class that is implemented on top of MPI (Message Passing Interface) and the MPICH libraries. Although much of this class's interface is similar to MPI, other communication techniques can be used in place of MPI. VTK also has communication classes based on shared-memory and sockets, either of which can be transparently substituted for MPI based class. On top of these low-level communication classes, VTK has also implemented remote-method invocations (RMIs), marshalling of VTK data sets, and ports to connect pipelines in different processes.

The multi-processing tools in VTK are very general and support a variety of parallel techniques, but for the application ParaView we only require VTK's data-parallel operation. In VTK's data-parallel mode, identical pipelines are created in every process, and each process is assigned a different piece of the data set. The pipelines can be connected through ports to collect the geometry in a single process for rendering, or the geometry can be rendered local to each process and the frame buffers collected with a sort-last compositing algorithm.

¹ VTK source and libraries can be downloaded from <http://www.kitware.com/vtkhtml/vtkdata/HowToGetSoftware.html>

Binary tree compositing is one simple parallel-rendering module that has been implemented in VTK. This module uses the first process to control interaction, while all other processes are waiting for messages to trigger RMI calls. RMIs change pipeline parameters and trigger remote renders/compositing. We chose this technique for ParaView's primary rendering mode because the large data remains distributed and scalability is virtually unlimited. Using parallel compositing and out-of-core streaming, VTK pipelines (containing iso-surface and probe filters) have been demonstrated on 900 terabytes of data distributed across 1024 processors^[2].

3 ParaView²

Although VTK is very powerful and can be easily scripted, it is only a toolkit. VTK can by itself process large data sets, but it is not easy to use. Most scientists are unwilling to write visualizations pipelines in C++ or Tcl. A turnkey application is needed to fully utilize VTK capabilities. We are creating an extendible open-source turn-key application called ParaView that is built on top of VTK and can process large data sets in parallel.

Besides usability and scalability, the most important design requirement for ParaView was extendibility. The application is open-source and we expected that others will contribute filters and rendering modules. The architecture had to be simple, so new features could be added easily. Any filter that is put into VTK can trivially be added to ParaView's user interface.

For extendibility, we separated the VTK processing engine from the user interface (UI). The user-interface module runs on the first process and is responsible for constructing and synchronizing the duplicate VTK pipelines in the satellite processes. Once this is achieved, VTK takes full control of the pipeline execution and distributed rendering. This division gives the application great flexibility. A pipeline created in ParaView can be saved and run separately with no dependence on the user interface or widget libraries. Such pipelines would be useful for batch processing, run-time visualization, or creating a visualization server. This separability and modular approach also means that the current widget set and UI architecture could be changed if necessary.

We took an innovative and simple approach to synchronizing the distributed pipeline with the user-interface module. Inter-process synchronization is handled almost exclusively with Tcl scripts. A Tcl interpreter, linked to VTK, is running on every process.

² Instructions for downloading and building ParaView can be found at: <http://www.kitware.com/ParaView>

This simplifies the pipeline control to a single method called "BroadcastScript(char*)". Since VTK is automatically wrapped in Tcl, no special effort was needed to create remote interfaces to the many VTK filters.

Unique Tcl command strings are created for each VTK-filter and data-object instance. These names are identical across all processes so they can be used as identity tags. Tcl handles hashing the names internally, so the application does not need to manage this task.

Adding new filters to the application is simple. After the filter has been added to VTK, it can easily be put in ParaView with a concise text description of its user interface. When ParaView starts, it reads a file that describes all the VTK filters that will be accessible from the application. The description is stored in an XML format shown in figure 1. Several widget options are available to map filter parameters. These include:

- String and numerical entries (multi-element)
- Sliders for clamped values
- Toggle buttons for Boolean values
- Option menus
- Data set selection menu
- File dialogs

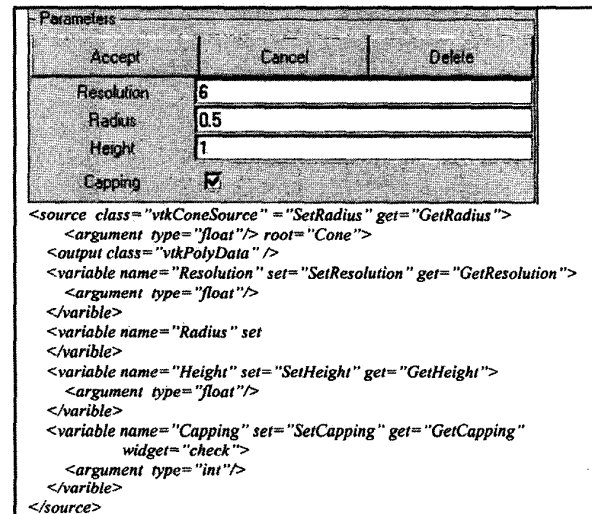


Figure 1: User interfaces for vtkFilters can be define in XML and loaded from a file. This interface is for VTK's cone source.

The resulting interfaces are rigid but they allow easy access to all of the filters and sources in VTK. To make the application easy to use, common filters have special UIs, which are accessible directly from icons on the toolbar. The following filters currently have custom interfaces:

- Iso Contouring
- Glyphing

- Cutting/Clipping
- Thresholding
- Attribute Calculator

Using the Tcl scripting language to control the distributed VTK pipeline was a natural choice because both the VTK pipeline objects and the user interface components are wrapped in Tcl. This approach also makes it extremely easy to add new components to the application. It allows simple text commands to define the connection between widgets and pipeline parameters.

3.1 Tcl and the Application Framework

The application framework is a unique blend of Tcl/Tk and C++. Tk is used as the widget set, but C++ objects are created to encapsulate the widgets and create higher-level UI components. Like VTK objects, these C++ UI objects are automatically wrapped in Tcl, with all public methods accessible from the scripting language. Tcl is used to communicate between the Tk widgets and the C++ objects, but is almost completely hidden from the developer. This approach takes advantage of the callback bindings of Tk, and keeps the object-oriented design and efficiency of C++.

3.2 Levels of Details

Although our parallel compositing module can handle any size data, the disadvantage of this approach is that rendering rates can be slow. Clusters that do not have graphics acceleration on every node must rely on software rendering. There is also a fixed cost for communicating the frame buffers to the first node for display. Although the frame buffers are relatively small, the communication occurs at every render.

To address these issues ParaView has two rendering modes: an interactive mode where detail is sacrificed for interactive frame rates, and a still mode, which occurs at the end of interaction. The still mode always uses tree compositing and may take several seconds. A few level-of-detail (LOD) techniques have been implemented to increase frame rates for interactive rendering. The combination of approaches selected depends on the specific capabilities of the cluster.

LOD techniques have been proven useful for interactively rendering other large models^[3,13]. It must be noted that no information is lost with this technique. Reduced resolution models are only temporarily substituted for the full resolution image while the user is interactively changing the view. If the user wants to see a full resolution image, they can simply stop the interaction (release the mouse button) to cause a render in still mode.

Sub-sampling is the first LOD technique we consider. Large rendering windows are often necessary to show all the detail of large data sets. When large windows are used, the communication of the color and Z buffers dominates over the rendering time. For interactive rendering, we render and composite smaller windows and use pixel replication to magnify the result for final display (see figure 3). This technique has also been used successfully for ray-cast volume rendering in VTK.

Rendering large geometry on each individual process can also be a bottleneck. This is especially true when using software rendering. In this situation, we use a distributed decimation algorithm^[5,6] that quickly generates small models to substitute for the original models. The chosen algorithm uses a quadric error measure and requires no communication between the processes. Table 1 compares the rendering times of a 1.8 million-triangle iso surface with different LOD techniques and on different number of processes.

	1 Proc.	2 Procs.	4 Procs.	8 Procs.	16 Procs.
Full Resolution	15.6 sec.	8.66 sec.	5.85 sec.	4.32 sec.	4.12 sec.
Decimated	0.43 sec.	1.02 sec.	1.46 sec.	1.84 sec.	2.63 sec.
Decimated and Subsampled	0.44 sec.	0.69 sec.	0.61 sec.	0.65 sec.	0.67 sec.

Table 2: Rendering rates for LOD techniques. The model was a 1.8 million-triangle iso-surface rendered in software with Mesa. The Decimated times are dominated by buffer access and communication for compositing. In the last row, the subsample rate was 3x3.

When both of these approaches fail to give adequate frame rates, the final solution is to transmit the simplified geometry to the first process where it can be rendered locally. This is the best LOD approach when hardware-rendering resources are available only to the first process.

4 Results

To demonstrate the application we used a POP ocean data set based on a 3200x2400x40 structured grid. The data has temperature, salinity and flow information for each element. The data set consumes ~6.3 Gbytes of disk, and expands to ~9.8 Gbytes of memory when the grid points and vertical flow component are reconstructed.

Figure 2 shows the application running on 16 processors of a distributed cluster with an ocean data set. An iso surface (0.001) of salinity was used for the ocean floor. The continents were generated with geometry filter and a clip filters. An iso surface of salinity at 0.0365 was taken and colored by ocean temperature. Figure 3 shows the same view when using LOD pixel subsampling rate of 3x3.

	1 Proc.	2 Procs.	4 Procs.	8 Procs.	16 Procs.
IsoSurface	8.25 sec.	4.12 sec.	2.23 sec.	1.23 sec.	0.75 sec.
Geometry	25.1 sec.	12.5 sec.	6.28 sec.	3.28 sec.	1.69 sec.
Clip	915 sec.	276 sec.	140 sec.	62.1 sec.	18.8 sec.
Decimation	————	119.2 sec.	59.8 sec.	29.6 sec.	15.1 sec.

Table 2: Scalable performance is demonstrated for three VTK filters. The Decimation filter reduced 17.3 million triangles to 14.8 thousand. I have no explanation for the super-linear performance of the clip filter.

Table 2 shows scalable performance for four different filters used in the pipeline from figure 2. A reduced resolution grid with 13 million cells was used to allow the pipeline to run efficiently on one process, but the decimation filter was tested on a different full resolution (3600x2400) model of the ocean surface.

Figure 4 shows a close up of the Straits of Gibraltar where the Mediterranean Sea meets the Atlantic Ocean. The 2D cut surface shows salinity values vs. depth. The ocean floor and continents were generated in the same way as figure 2.

Figure 5 shows flow in the Atlantic Ocean near the Mediterranean Sea. After the ocean floor and continents were extracted, the number of sample points was reduced with a threshold filter that eliminated points with small vector magnitudes. The vectors were further reduced with a spatial-exclusion filter. Vectors inhibited neighboring samples so that each vector had a sphere with no other samples in it. The radii of these spheres were scaled by the vector magnitude. A glyph filter was then used to display the vectors as arrows, which were scaled and colored by the vector magnitude.

5 Future Work

Although VTK sources and filters can easily be added to ParaView's user interface, not all VTK filters can run in a distributed environment or produce piece-invariant results. VTK has implemented ghost cells for filters that require information about neighboring cells, and a few dozen filters in VTK already run in a distributed piece-invariant manner. However, more work is required to create alternatives in VTK for filters that cannot be easily converted. Connectivity, multi-pass smoothing, and streamlines are a few examples of such filters.

Also, the distributed rendering module currently used in ParaView is a rather simple tree-composite algorithm. More efficient alternatives have to be implemented. A couple options include: binary swap compositing^[11], or an algorithm that compresses the buffers to reduce communication costs^[1]. Integrating a library for rendering on large tiled displays^[7], would also be useful.

6 Conclusion

In this paper we presented a scalable application architecture that is based on data parallelism in VTK and has been developed to handle extremely large data sets. The use of Tcl interpreters to synchronize the distributed pipelines simplifies the application and the addition of new features. The open source application "ParaView" has been developed using the architecture and was demonstrated on a 6.9 GB ocean temperature and salinity data set.

6 Acknowledgements

This work was supported in part by grants from the US Department of Energy ASCI Views program and the DOE Office of Science.

The ocean model³ was supplied by Mathew Maltrud from Los Alamos National Laboratory.

7 References

- [1] Garland, Michael and Paul S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *Proceedings of IEEE Visualization '98*, pages 263 – 269, 1998.
- [2] Garland, Michael and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. In *Proceedings of ACM SIGGRAPH '97*, pages 209 – 216, 1997.
- [3] Heckbert, Paul S. and Michael Garland. Survey of Polygonal Surface Simplification Algorithms. *ACM SIGGRAPH '97 Course Notes*, 1997.
- [4] Hoppe, Hughes and Steve Marschner. Efficient Minimization of New Quadric Metric for Simplifying Meshes with Appearance Attributes. (Adendum to IEEE Visualization 1999 paper) *Microsoft Research Technical Report MSR-TR-2000-64*, June 2000.
- [5] Hoppe, Hughes. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *Proceedings of IEEE Visualization '99*, pages 59 – 66, 1999.
- [6] Lindstrom, Peter. Out-of-Core Simplification of Large Polygonal Models. In *Proceedings of ACM SIGGRAPH 2000*, pages 259 – 262, 2000.
- [7] Low, Kok-Lim and Tiow-Seng Tan. Model Simplification Using Vertex-Clustering. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 75 – 81, 1997.
- [8] Rossignac, Jarek and Paul Borrel. Multi-Resolution 3D Approximations for Rendering Complex Scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455-465, 1993.
- [9] Schroeder, William J., Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. In *Proceedings of ACM SIGGRAPH '92*, pages 65 – 70, 1992.

³ Data sets from POP simulations can be downloaded from <http://www.acl.lanl.gov/climate/>