VisBench: A Framework for Remote Data Visualization and Analysis

Randy W. Heiland, M. Pauline Baker, and Danesh K. Tafti

NCSA, University of Illinois at Urbana-Champaign,
Urbana, Illinois
{heiland, baker, dtafti}@ncsa.uiuc.edu

Abstract. Computational researchers typically work by accessing a compute resource miles from their desk. Similarly, their simulation output or other data is stored in remote terabyte data servers. VisBench is a component-based system for visualizing and analyzing this remote data. A time-varying CFD simulation of heat exchange over a louvered fin provides sample data to demonstrate a workbench-oriented version of VisBench. An analysis technique (POD) for spatiotemporal data is described and applied to the CFD data.

1 Introduction

Computational and experimental scientists routinely access remote resources necessary for their research. While it was once customary for a researcher to download remote data to a local workstation in order to visualize and analyze it, this scenario is usually impractical today. Data files have become too large and too numerous from a typical simulation run at a high-performance computing center. This has led to data being stored in terabyte servers located at the HPC sites.

Experimental scientists face the same dilemma of burgeoning remote data. For example, the Sloan Digital Sky Survey has already captured nearly two years worth of data and the Large Hadron Collider at CERN and National Ignition Facility at LLNL will each produce vast amounts of data when they become fully operational.

The goal of the NSF PACI program is to build the Grid[1] – a distributed, metacomputing environment connected via high-speed networks, along with the necessary software to make it usable by researchers. As part of this effort, we present a software framework that is being used to remotely visualize and analyze data stored at NCSA.

Some goals of our project, called *VisBench*, include:

- minimize data movement,
- take advantage of (remote) HPC resources for visualization and analysis,
- provide application-specific *workbench* interfaces and offer a choice of clients, ranging from lightweight (run anywhere) to more specialized,
- be prudent of the cost of necessary software.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 718–727, 2001.

The notion of an application-specific workbench means offering high-level functionality that is pertinent to an application domain. We will demonstrate this with an analysis technique that VisBench offers through its CFD interface. The last goal stems from the fact that, as leading-edge site of the Alliance[1], NCSA will make software recommendations to our partners.

We now present an overview of VisBench, followed by an example of it being used to visualize and analyze remote CFD data.

2 VisBench: A Component-Based Workbench

VisBench adopts the software component model whereby application components are connected in a distributed object fashion. This model allows us to "plug-in" various software components – taking advantage of their individual strengths. A component may be anything from a small piece of code that performs one specific function to an entire software package that provides extensive functionality.

We have initially taken a coarse-grained approach; our framework consists of just a few, large components. In this paper we will present just three: a general-purpose visualization component, a general-purpose data analysis component, and a graphical user interface component. For visualization, we use an open source software package called the Visualization Toolkit (VTK)[2]. VTK is a relative newcomer in the visualization community and is being rapidly adopted by many groups. For data analysis, we use MATLAB, a commercial software package that has been in existence for over twenty years. For the user interface, we have written a graphical user interface (GUI) client using Java Swing.

In order to connect these different components together and have them easily interoperate, one needs *middleware*. We have initially chosen CORBA as the middleware for VisBench. Figure 1 shows a schematic of the VisBench framework discussed in this paper.

2.1 Visualization

We selected VTK as our core visualization component for a number of reasons. As a modern object-oriented software package (written in C++), it offers over 600 classes that perform numerous 2-D and 3-D visualization algorithms, rendering, and interaction techniques. The quality of example renderings, especially for scientific datasets, and the ability to interact in 3-D were important factors. So too was its ability to stream very large datasets through various filters within a visualization pipeline. Its open source model allows for valuable contributions from users. For example, a group at LANL has provided extensions for parallel filters using MPI. The Stanford Graphics group will be offering a parallel rendering version of VTK, and the Vis&VE group at NCSA has provided functionality for easily using VTK in the CAVE – a future VisBench client. The level of activity and excitement within the VTK community is very encouraging.

[1] National Computational Science Alliance, one of two partnerships within PACI

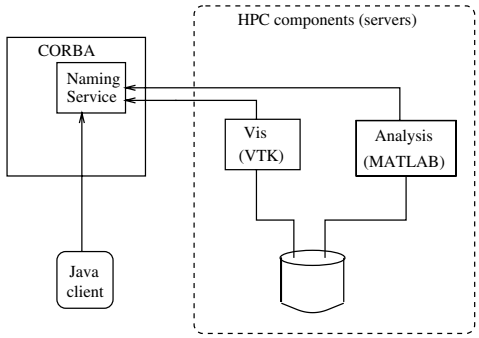

Fig. 1. VisBench design and some existing components

One feature lacking in the VTK distribution is a graphical user interface. There is, however, an option to automatically wrap each class in the Python or Tcl scripting language. Our visualization component (server) consists of an embedded Python interpreter that has access to all the VTK classes, providing a mechanism for dynamically constructing a visualization pipeline. A VisBench user does not need to know the Python language though. The Java GUI (client) transparently converts all actions into appropriate Python-VTK commands. This has the added benefit of generating a script, associated with an interactive session, which can then be run in batch mode if necessary.

2.2 Data Analysis

The MATLAB software package is familiar to nearly every scientist and engineer who performs numerical calculations. It has earned wide respect and is trusted for its numerical accuracy. Using its Application Program Interface, we have written a core analysis component for VisBench. As with the visualization component, the intent is to provide high level point-and-click workbench-related functionality. Of course, if users are capable of writing their own MATLAB scripts, these too can be entered via VisBench and executed on the remote analysis server.

Besides providing extensive core mathematical functionality, MATLAB has another distinct advantage over similar systems – there are numerous, freely available scripts that cover nearly every application domain. It is straightforward to incorporate such scripts into VisBench.

2.3 User Interface

In order to make the remote VisBench engines accessible to users working on a wide variety of platforms, our primary VisBench client is written in Java. The client makes extensive use of the Java Swing package for the graphical user interface. A sample VisBench session would proceed as follows:

- from the client, connect to a running server (a factory server that forks off a separate server for each user),
- using a Reader widget, read a (remote) data file; associated metadata will be displayed in the widget,
- create filters for the data – for example, slicing planes, isosurfaces, streamlines, vector glyphs, etc.,
- geometry associated with the filters is rendered on the back-end VTK server and the resulting image compressed and sent back to the client where it gets displayed,
- interactively view the results and make necessary changes.

An example of the CFD workbench interface is shown in Figure 2. This Java client is a very lightweight client. One needs only a Java Virtual Machine in order to run it. (The Java 2 platform is required since we use the Swing package and the CORBA bindings).

Workbench interfaces are being designed as a collaborative effort between application scientists and computer scientists. We envision these workbenches evolving into problem-solving environments.

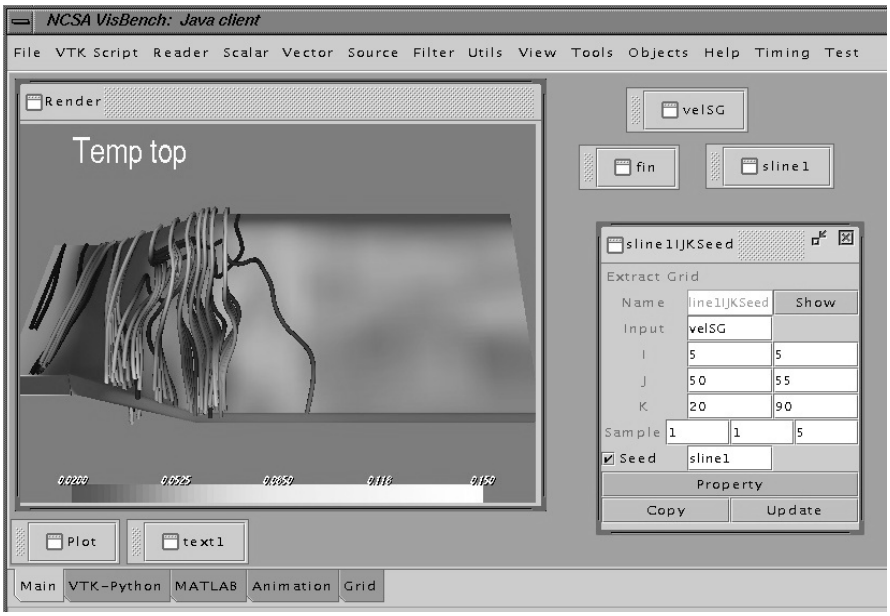

Fig. 2. An example of the CFD Java client

2.4 Middleware

Middleware is the software that allows distributed objects to communicate and exchange data with each other. There are three mainstream middleware so-

lutions: COM, CORBA, and JavaBeans. COM (Component Object Model) is from Microsoft and is intended for Windows platforms and applications. CORBA (Common Object Request Broker Architecture) is a vendor-independent specification defined by a not-for-profit consortium, the Object Management Group. JavaBeans, from Sun, is for Java platforms and applications.

We chose CORBA as the middleware for VisBench for two primary reasons. It provides language interoperability – something quite valuable in scientific computing where there is a mix of Fortran, C/C++, and, with growing frequency, Java. It makes the most sense based on our current high-end hardware architectures and operating systems – primarily IRIX and Linux. We selected the ACE ORB (TAO), an open source university research project, as our CORBA implementation. An Object Request Broker (ORB) provides a mechanism for transparently communicating requests from a client to a *servant object*. Within the scope of VisBench that is being presented here, there are only two servant objects – the VTK and MATLAB components.

The interfaces to CORBA objects are specified using the Interface Definition Language (IDL). The IDL is independent of any programming language and allows for applications in different languages to interoperate. The IDL for a particular object contains the interfaces (methods) for its operations.

CORBA provides the specifications for a number of different *services*. The most commonly used is the Naming Service. The Naming Service provides a mechanism for mapping object names to object references. A servant object "advertises" itself via the Naming Service. A client then connects to the Naming Service, obtains object references, and invokes operations on those objects.

VisBench has less communication overhead than other HPC distributed component models. In the Common Component Architecture[3], for example, the goal is to construct a distributed computational pipeline requiring the transmission of large amounts of data. The current VisBench model assumes that the simulation (or experimental) data will be directly accessible from the machine hosting the VisBench server and that a relatively small amount of visualization, analysis, or user data will be exchanged between client and server.

3 POD Analysis

When trying to understand spatiotemporal data, visualization – especially an animation, is an extremely useful tool. However, sometimes one needs more quantitative information – for example, when trying to compare results from a parameter study.

We present a fairly complex analysis technique that is sometimes used in CFD research – particularly in the study of turbulence. The technique has its roots in statistics and is based on second-order statistical properties that result in a set of optimal eigenfunctions. In certain application domains, the algorithm is known as the Karhunen-Loève decomposition. Lumley[4] pioneered the idea that it be used to analyze data from turbulent flow simulations and suggested that it could provide an unbiased identification of coherent spatial structures. Within

this context, the algorithm is known as the *proper orthogonal decomposition* (POD).

We use the POD as an example of providing high-level analysis functionality in the VisBench environment. Because of its computational demands, the POD is most often used for analyzing only 1-D or 2-D time-varying data. Since VisBench provides an analysis component running on a remote HPC resource, we will demonstrate the POD on time-varying 3-D data. An outline of the algorithm is presented.

Assume we are given a set of numeric vectors (real or complex):

$$\{\mathbf{X}_i\}_{i=1}^{M}$$

where $\mathbf{X} = [x_1, x_2, \ldots, x_N]$.

The mean is computed as:

$$\overline{\mathbf{X}} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{X}_i$$

Hence, if we have a time series of spatial vectors, the mean will be the time average.

We shall operate on a set of caricature vectors with zero mean:

$$\hat{\mathbf{X}}_i = \mathbf{X}_i - \overline{\mathbf{X}}, \qquad i = 1, \ldots, M$$

Using the method of snapshots[5], we construct an approximation to a statistical covariance matrix:

$$\mathbf{C}_{ij} = \langle \hat{\mathbf{X}}_i, \hat{\mathbf{X}}_j \rangle, \qquad i, j = 1, \ldots, M$$

where $\langle \cdot, \cdot \rangle$ denotes the usual Euclidean inner product.

We then decompose this $M \times M$ symmetric matrix, computing its (non-negative) eigenvalues, λ_i, and its eigenvectors, $\boldsymbol{\phi_i}$, $i = 1, \ldots, M$, which form a complete orthogonal set.

The orthogonal eigenfunctions of the data are defined as:

$$\Psi^{[k]} = \sum_{i=1}^{M} \phi_i^{[k]} \hat{\mathbf{X}}_i, \qquad k = 1, \ldots, M$$

where $\phi_i^{[k]}$ is the i-th component of the k-th eigenvector. It is these eigenfunctions which Lumley refers to as *coherent structures* within turbulent flow data.

The *energy* of the data is defined as the sum of the eigenvalues of the covariance matrix:

$$E = \sum_{i=1}^{M} \lambda_i$$

Taking the ratio of an eigenvalue (associated with an eigenfunction) to the total energy, we calculate an energy percentage for each eigenfunction.

$$E_k = \frac{\lambda_k}{E}$$

Sorting the eigenvalues (and associated eigenvectors) from largest to smallest, we can order the eigenfunctions from most to least energetic. In a data mining context, we refer to the plot of descending eigenfunction energy percentages as a *quality of discovery* plot. Ideally, one wants just a few eigenfunctions that cumulatively contain most of the energy of the dataset. This would be an indication that these eigenfunctions, or coherent structures, are a good characterization of the overall data. (In a dynamical systems context, this would constitute a relatively low-dimensional phase space of the given system).

Furthermore, since the eigenfunctions span the space of the given data, it is possible to reconstruct an approximation to any given vector as:

$$\mathbf{X} \approx \overline{\mathbf{X}} + \sum_{i=1}^{K} a_i \Psi^{[i]}$$

taking the first $K (K < M)$ most energetic eigenfunctions, where the coefficients are computed by projecting the caricature vectors onto the eigenfunctions:

$$a_i = \left(\frac{\hat{\mathbf{X}} \cdot \Psi^{[i]}}{\Psi^{[i]} \cdot \Psi^{[i]}} \right)$$

The POD has been used to analyze near wall turbulence and other PDE simulations[6][7], as well as experimental data.

4 Example: Turbulent Flow Over Fins

A HPC simulation of turbulent flow and heat transfer over a multilouvered fin[8] provides a source of example data for demonstrating VisBench. Figure 3a shows the multilouvered fin model. The flow is from left-to-right, i.e., hitting the leading (upper) edge of a fin and traveling down to the trailing edge. For the simulation, the computational domain is comprised of one-half of one full fin (due to symmetry) and the region around it, as shown in Figure 3b. It is periodic back-to-front (leading-edge boundary to trailing-edge boundary) and bottom-to-top. There is a wall at the flat base where much of the heat is retained.

The computational domain consists of a topologically regular grid which follows the geometry of the fin – having higher resolution in the area of interest (at the twisted junction). The grid size is $98 \times 98 \times 96$ for this particular model.

The simulation writes files of computed values at regular time intervals. From the VisBench client, a user selects some subset (M) of these remote files, representing a particular regime of the simulation. The user then selects a particular scalar or vector field within each file and, after connecting to an analysis server,

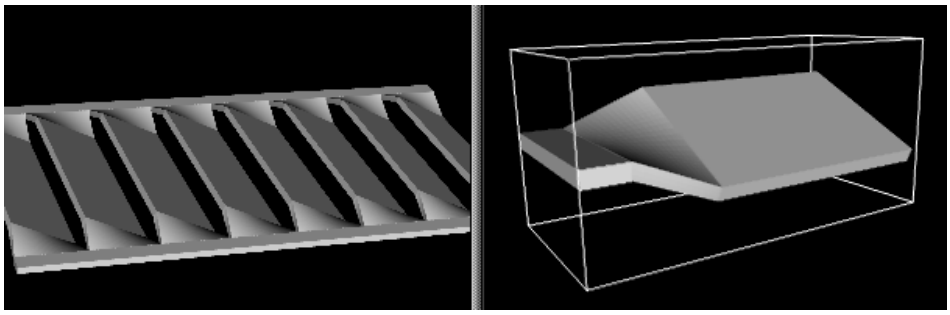

Fig. 3. a) The multilouvered fin and b) the computational domain

initiates the POD analysis. We have currently analyzed the temperature, pressure, streamwise vorticity, and velocity fields.

Figure 4 shows the results of applying the POD to the temperature field (for $M \approx 200$). The mean is shown at the top followed by the first three most energetic eigenfunctions. The first eigenfunction contained 32% of the energy, the second, 12%, and the third, 8%.

The first eigenfunction reveals "hot-spots" along the flat base (top and bottom), a circular region near the upper part of the top, and two streaks along the length of the bottom of the fin. The second eigenfunction visually appears to be orthogonal to the first (which it is, mathematically) – the base of the fin is now "cool" and there is a hot-spot near the upper region of the twisted junction (both top and bottom). Although we show only the surface of the fin (color-mapped with temperature), it should be noted that the analysis was indeed performed over the entire 3-D domain and has been visualized using various filters.

By encapsulating the POD algorithm in the analysis component, interactive viewing from the visualization component, and relevant graphical controls from a client, we have illustrated how VisBench can serve as an application workbench.

5 Summary

We have presented a framework for performing visualization and analysis of remote data. The VisBench model consists of a variety of servers running on remote HPC machines with access to data archived on those machines. An application-dependent (workbench) client with a graphical user interface can simultaneously communicate with different servers via CORBA. We have shown a Java client used for visualizing and analyzing CFD data. The client is lightweight in that it requires only the Java 2 runtime environment. The amount of data exchanged between client and visualization server is minimal – GUI-generated command strings to the server and compressed (JPEG) images to the client.

An earlier version of VisBench was demonstrated at Supercomputing '99 – a Java client on the show floor in Portland visualizing CFD data at NCSA. In addition, VisBench and Condor (`www.cs.wisc.edu/condor`) teamed together to

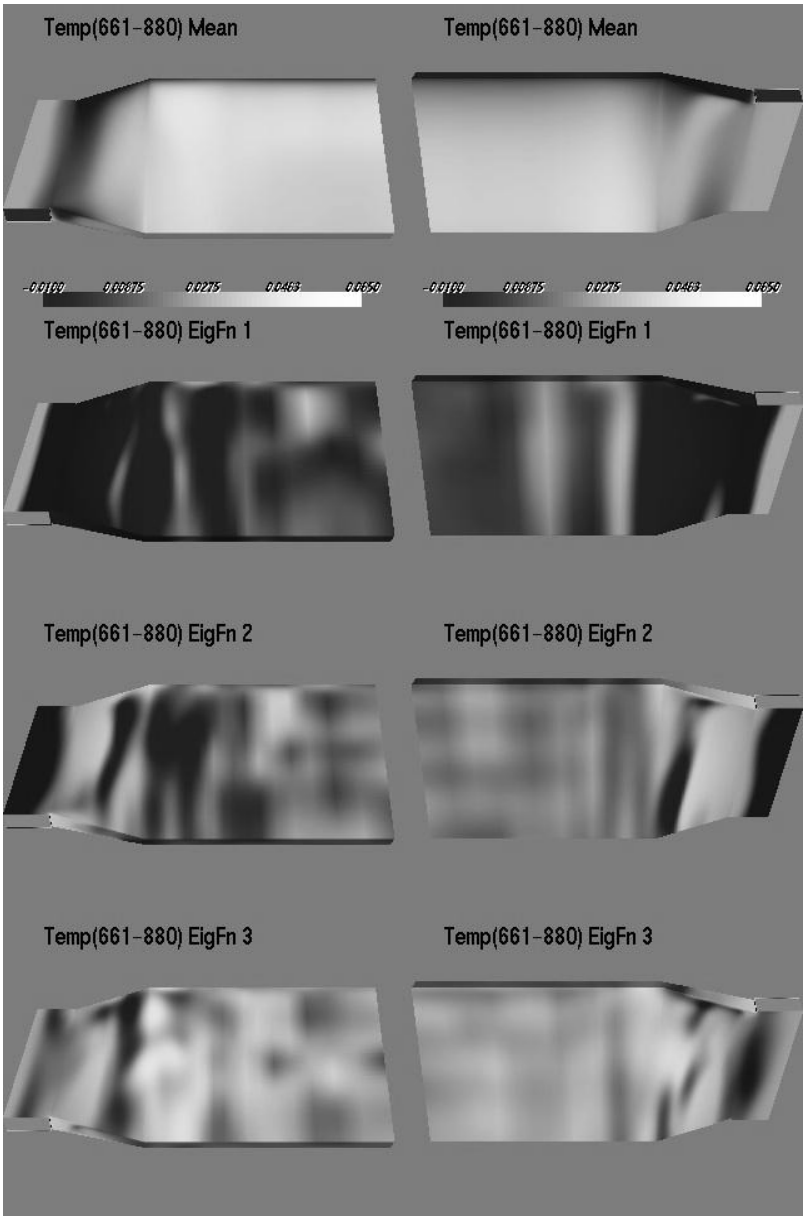

Fig. 4. POD results of temperature on fin (top and bottom)

demonstrate frame-based parallel rendering using the Grid. At Supercomputing '00 in Dallas, we demonstrated both the Java client (for a Chemical Engineering workbench) and a geometry client that received geometry from a VisBench server (at NCSA) and displayed it locally on a tiled wall.

We have not yet collected performance data for the different scenarios that VisBench offers. The visualization server that typically runs at NCSA (a multiprocessor SGI Onyx) takes advantage of hardware rendering and has shown a 50x speedup over software rendering. This is highly desirable for an image-based client, such as the Java client presented here or a client in a web browser. To offer an idea of performance for the Java client, the rendering in Figure 2 would update at about 2 frames/second at the Supercomputing conferences. For a geometry client, the rendering speed will be determined by the client machine's hardware. Obviously, VisBench will always be network limited. We have not yet incorporated parallel VTK filters into the visualization server, nor are we using parallel rendering. These two areas will receive more attention as we transition to Linux clusters. The analysis component is still considered proof of concept and the POD implementation, in particular, needs to be optimized.

Much of our effort has gone toward providing utilities that the application scientists have requested – remote file selection, animation, camera path editing, and providing VRML files on the server machine (which doubles as a web server).

We acknowledge the help of many members, past and present, of the NCSA Vis&VE group, as well as members of the Condor team, the Alliance Chemical Engineering team, and the Extreme! Computing team at Indiana University. For more information on VisBench, we refer readers to the web page at `visbench.ncsa.uiuc.edu`.

References

1. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufmann, San Francisco (1998)
2. Schroeder, W., Martin, K., Lorensen, W.: The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 2nd ed.. Prentice-Hall, Old Tappan, N.J. (1998)
3. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B.: Toward a Common Component Architecture for High-Performance Scientific Computing. Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (1999) 115-124
4. Lumley, J.L.: The structure of inhomogeneous turbulent flow. In: Yaglom, A.M., Tatarski, V.I. (eds.): Atmospheric Turbulence and Radio Wave Propagation **25** (1993) 539-575
5. Sirovich, L.: Turbulence and the dynamics of coherent structures: Part I-III. Quarterly of Applied Mathematics, XLV(3) (1987) 561-590
6. Aubry, N., Holmes, P., Lumley, J.L., Stone, E.: The dynamics of coherent structures in the wall region of a turbulent boundary layer. J. Fluid Mech. **192** (1988) 115-173
7. Berkooz, G., Holmes, P., Lumley, J.L.: The proper orthogonal decomposition in the analysis of turbulent flows. Annual Review of Fluid Mechanics **25** (1993) 539-575
8. Tafti, D.K., Zhang, X., Huang, W., Wang, G.: Large-Eddy Simulations of Flow and Heat Transfer in Complex Three-Dimensional Multilouvered Fins. Proceedings of FEDSM2000: 2000 ASME Fluids Engineering Division Summer Meeting (2000) 1-18