FEDSM2003-45196

VISUALIZING COMPUTATIONAL SIMULATION RESULTS USING VIRTUAL REALITY TECHNOLOGY

Nilay Sezer-Uzol
Department of Aerospace Engineering
The Pennsylvania State University
University Park, PA

Anirudh Modi
Intel Corporation
Hillsboro, Oregon

Lyle N. Long
Department of Aerospace Engineering
The Pennsylvania State University
University Park, PA

Paul E. Plassmann
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA

ABSTRACT

The visualization of computational simulations of complex physical problems using virtual reality technology is demonstrated in this study. A general-purpose computational steering system (POSSE) which can be coupled to any C/C++ simulation code, has been developed and tested with a 3-D parallel Navier-Stokes flow solver (PUMA2) [1]. In addition, the visualizations can be displayed using virtual reality facilities (such as CAVEs and RAVEs) to better understand the 3-D nature of the flowfields. The simulations can be run on parallel computers such as Beowulf clusters, while the visualization is performed on other computers, through a client-server approach. A key advantage of our system is its scalability. Visualization primitives are generated on the parallel computer. This is essential for large-scale simulations, since it is often not possible to post-process the entire flowfield on a single computer due to memory and speed constraints. Example applications of time-dependent and three-dimensional computational flow simulations performed at Penn-State are presented to show the usefulness of POSSE and virtual reality systems. The examples include CFD predictions for unsteady simulations of a helicopter rotor, unsteady ship airwake simulations, helicopter tail fan-in-fin flow simulations and simulations of time-accurate flow and noise due to a landing gear.

INTRODUCTION

Parallel simulations are playing an increasingly important role in all areas of science and engineering. As the applications for these simulations expand, the demand for their flexibility and utility grows. Interactive computational steering is one way to increase the utility of these high-performance simulations, as they facilitate the process of scientific discovery by allowing the scientists to interact with their data. On yet another front, the rapidly increasing power of computers and hardware rendering systems has motivated the creation of visually rich and perceptually realistic Virtual Environment (VE) applications. The combination of the two provides one of the most realistic and powerful simulation tools available to the scientific community.

While a tremendous amount of work has gone into developing Computational Fluid Dynamics (CFD) software, little has been done to develop computational steering tools that can be directly integrated with such CFD software. Without such computational steering systems, the immediate and direct interaction with simulations is impossible. Instead, visualization of these simulations is mainly limited to the post-processing of data. At Penn State an easy to use, general-purpose, computational steering library: *Portable Object-oriented Scientific Steering Environment (POSSE),* has been developed that can be easily coupled to any existing C/C++ simulation code to address this need.

SOFTWARE FOR VIRTUAL REALITY SYSTEMS

The software architecture that we consider in this paper (illustrated in Figure 1) consists of two software components. The CFD simulation, as the generator of simulation data, must have the capacity to act as a data server to remote clients. These remote clients can interact with the simulation data in a passive manner (such as a visualization client) or in an active role (such as a steering client that changes the state of the simulation as it is executing). The POSSE system developed at Penn State can support both the data server and any number of clients.

This steering system has the ability to be integrated with a wide variety of visualization clients capable of supporting VE displays and devices. Today there is a variety of commercial and academic software packages that can be used. One of the most widely used VE display systems is the CAVE and RAVE systems now marketed by FakeSpace, but originally developed at the Electronics Visualization Laboratory [2]. The software library to support these devices is called CAVElibs and is commercially supported from VRCO.

Recently, a great deal of work has been done to develop more portable software systems to support VE devices and displays. These newer systems are academic, not commercially supported, but are more modular in their software implementation and offer users a greater degree of flexibility in their use. For example, the Device Independent Virtual Environments, Reconfigurable, Scalable, Extensible (DIVERSE) is a collection of software modules that can be used to implement visualization clients, as an API, or to integrate heterogeneous VR environments [3]. VRjuggler is a software library that has been developed to support a wide variety of VE devices and has been ported to variety of architectures [4]. Virtual Environments are also now more commonly being supported by commercial visualization systems. For example, Ensight now supports stereo displays and some VE devices. This range of software options allows today's users a much greater degree of flexibility both in terms of software sophistication and the effort required to develop a VE visualization client. Any of these VE software systems can be used in conjunction with POSSE.

COMPUTATIONAL MONITORING AND STEERING LIBRARY: POSSE

The computational monitoring and steering library, POSSE [1, 5, 6], is written in C++, using advanced object-oriented features, making it powerful, while maintaining the ease-of-use by hiding most of the complexities from the user. The library allows a simulation running on any parallel or serial computer to be monitored and steered remotely from any machine on the network using a simple cross-platform client utility. This library has been used to augment the parallel flow solver, Parallel Unstructured Maritime Aerodynamics-2 (PUMA2), which is written in C using the Message Passing Interface (MPI) library, to obtain a powerful interactive CFD system. This system is being successfully used to monitor and steer several large flow simulations over helicopter, ship, and other geometries. This approach provides the user with a fast and simple debugging and analysis mechanism, where the flow and convergence parameters can be changed dynamically without having to kill or restart the simulation. This CFD system, which primarily runs on an in-house Beowulf Cluster, the COst-effective COmputing Array-2 (COCOA-2) [7, 8, 9], has been coupled to our Virtual Reality (VR) system, a Fakespace Reconfigurable Automatic Virtual Environment (RAVE) [10], to obtain near real-time visualization of the 3-D solution data in stereoscopic mode. This ability to get "immersed" in the complex flow solution as it unfolds using the depth cue of the stereoscopic display and the real-time nature of the computational steering system opens a whole new dimension to the engineers and scientists interacting with their simulations.

While running a complex parallel program on a high-performance computing system, one often experiences several major difficulties in observing computed results. Usually, the simulation severely limits the interaction with the observing program during the execution and makes the visualization and monitoring slow and cumbersome (if at all possible), especially if it needs to be carried out on a different system (say a specialized graphics workstation for visualization).

For CFD simulations, it is very important for the surface contours of flow variables to be computed instantaneously and sent to the visualization client in order for the user to observe it and take appropriate action. This activity is referred to as "monitoring," which is defined as the observation of a program's behavior at specified intervals of time during its execution. On the other hand, the flow variables and/or solver parameters may needed to be modified as the solution progresses. Thus, there is a need to modify the simulation based on these factors by manipulating some key characteristics of its algorithm. This activity is referred to as "steering," which is defined as the modification of a program's behavior during its execution.

Software tools which support these activities are called computational steering environments. These environments typically operate in three phases: instrumentation, monitoring, and steering. Instrumentation is the phase where the application code is modified to add monitoring functionality. The monitoring phase requires the program to run with some initial input data, the output of which is observed by retrieving important data about the program's state change. Analysis of this data gives more knowledge about the program's activity. During the steering phase, the user modifies the program's behavior (by modifying the input) based on the knowledge gained during the previous phase by applying steering commands, which are injected on-line, so that the application need not be stopped and restarted.

The steering software, POSSE, is very general in nature and is based on a simple client/server model. It uses an approach similar to Falcon [11] (an on-line monitoring and steering toolkit developed at Georgia Tech) and ALICE Memory Snooper [12] (an application programming interface designed to help in writing computational steering, monitoring and debugging tools developed at Argonne National Lab). Falcon was one of the first systems to use the idea of threads and shared memory to serve program data efficiently. POSSE consists of a steering server on the target machine that performs steering, and a steering client that provides the user interface and control facilities remotely (Figure 1). The steering server is created as a separate execution thread of the application to which local monitors forward only those registered data (desired program variables, arrays and/or structures) that are of interest to steering activities. A steering client receives the application run-time information from the server, displays this information to the user, accepts steering commands from the user, and enacts changes that affect the application's execution. Communication between a steering client and server is done via UNIX sockets and threading is done using POSIX (Portable Operating System Interface standard) threads. POSSE has been written completely in C++, using several of C++'s advanced object-oriented features, making it fast and powerful, while hiding most of the complexities from the user.

POSSE is designed to be extremely lightweight, portable (runs on all Win32 and POSIX compliant Unix platforms) and efficient. It deals with byte-ordering and byte-alignment

problems internally and also provides an easy way to handle user-defined classes and data structures. It is also multi-threaded, supporting several clients simultaneously. It can also be easily incorporated into parallel simulations based on the Message Passing Interface (MPI) [13] library. The biggest enhancement of POSSE over existing steering systems is that it is equally powerful, yet extremely easy to use, making augmentation of any existing C/C++ simulation code possible in a matter of hours. It makes extensive use of C++ classes, templates and polymorphism to keep the user Application Programming Interface (API) elegant and simple to use. Due to its efficient design, POSSE has low computational overhead (averaging less than 1% relative to the computation thread) making it lightweight.

Figure 1 shows a schematic view of how POSSE can be used, an on-going scientific simulation is running on a remote Beowulf computing cluster. Any number of remote clients can query/steer registered data simultaneously from the simulation via the DataServer thread. Two clients are shown, a visualization client and a Graphical User Interface (GUI) client that provides a simple user interface to all registered simulation data. The visualization code can be used to interactively monitor a dataset at various time intervals and the GUI code can be used to steer the simulation by changing certain parameters associated with it. The client is written in C++ with the cross-platform FLTK [14] API for the GUI and the Visualization ToolKit (VTK) [15] for the 3-D visualization. Since VTK is written on top of OpenGL, the resulting application can benefit from the OpenGL hardware acceleration available on most modern-graphics chipsets. VTK also supports stereographics, and can thus be used in conjunction with the RAVE.
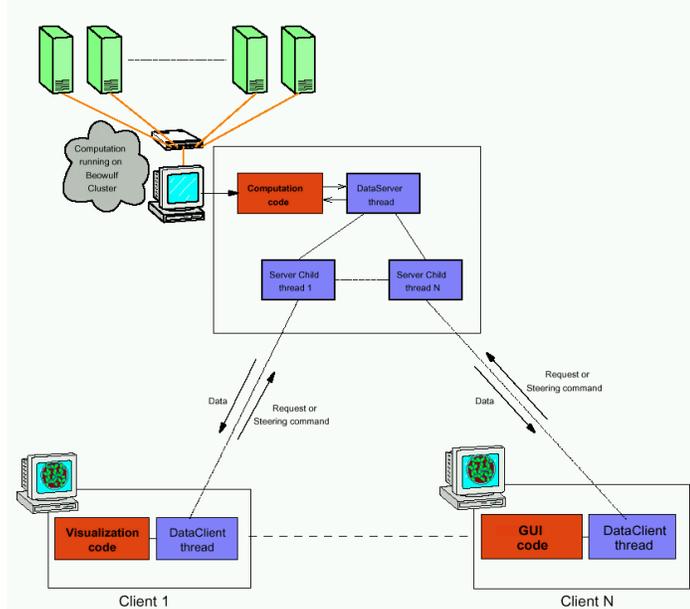

Figure 1. Schematic view of POSSE

Figure 2 and Figure 3 illustrate a simple, yet complete, POSSE client/server program in C++. As seen in the figures, registered data on the steering server (which are marked *read-write*) are protected using binary semaphores when they are being updated in the computational code. User-defined data

structures are handled by a simple user-supplied pack and unpack subroutine that call POSSE data-packing functions to tackle the byte-ordering and byte-alignment issues. The programmer need not know anything about the internals of threads, sockets or networking in order to use POSSE effectively.

Among other applications, POSSE has been successfully used to visualize a wake-vortex simulation of several aircraft in real-time [6, 16].

```
//--------------SERVER----------------

#include "dataserver.h"

int dummyInt = 0, n1, n2;
double **dyn2D;

REGISTER_DATA_BLOCK() // Register global data
{
    // Read-write data
    REGISTER_VARIABLE("testvar", "rw", dummyInt);
    // Read-only data
    REGISTER_DYNAMIC_2D_ARRAY("dyn2D", "ro", dyn2D, n1, n2);
}

int main(int argc, char *argv[])
{
    DataServer *server = new DataServer;

    // Start Server thread
    if (server->Start(4096) != POSSE_SUCCESS) {
        delete server;
        exit(-1);
    }
    n1 = 30; n2 = 40;
    ALLOC2D(&dyn2D, n1, n2);

    for (int iter = 0; iter < MAX_ITER; iter++) {
        // Lock DataServer access for dyn2D
        server->Wait("dyn2D");
        // Update dyn2D with new values
        Compute(dyn2D);
        // Unlock DataServer access for dyn2D
        server->Post("dyn2D");
    }
    FREE2D(&dyn2D, n1, n2);
    delete server;
}
```

Figure 2. A simple, complete POSSE server application written in C++

```
//--------------CLIENT----------------
#include "dataclient.h"

int main(int argc, char *argv[])
{
    DataClient *client = new DataClient;
    double **dyn2D;

    // Connect to DataServer
    if (client->Connect("cocoa.ihpca.psu.edu",4096)
!= POSSE_SUCCESS) {
        delete client;
        exit(-1);
    }
    // Send new   value for "testvar"
    client->SendVariable("testvar", 100);
    int n1 = client->getArrayDim("dyn2D", 1);
    int n2 = client->getArrayDim("dyn2D", 2);
    ALLOC2D(&dyn2D, n1, n2);
    client->RecvArray2D("dyn2D", dyn2D);
    Use(dyn2D); // Utilize dyn2D
    FREE2D(&dyn2D, n1, n2);
    delete client;
}
```

Figure 3. A simple, complete POSSE client application written in C++

3

PARALLEL CFD FLOW SOLVER: PUMA2

PUMA2 (The Parallel Unstructured Maritime Aerodynamics-2) is a parallel unstructured 3-D CFD flow solver [1, 6, 17, 18, 19]. It uses a finite volume formulation of the Euler/Navier-Stokes equations for 3-D, internal and external, non-reacting, compressible solutions of problems for complex geometries. PUMA2 can be run so as to preserve time accuracy or as a pseudo-unsteady formulation to enhance convergence to steady state. Mixed topology unstructured grids composed of tetrahedra, wedges, pyramids and hexahedra are supported in PUMA2. Different time integration algorithms such as Runge-Kutta, Jacobi and various Successive Over-Relaxation Schemes (SOR) are also implemented. It is written in ANSI C using the MPI library for message passing so it can be run on parallel computers and clusters. It is also compatible with C++ compilers and coupled with the computational steering system POSSE. It uses dynamic memory allocation, thus the problem size is limited only by the amount of memory available on the machine. Large eddy simulations can also be performed with PUMA2. Penn State has been refining and developing this code since 1997.

COMPUTATIONAL SIMULATIONS

Example applications of time-dependent and three-dimensional flow simulations performed at Penn-State are discussed below.

Helicopter Rotor Flow Simulations using CFD

The near flowfield of rotorcraft and tiltrotor wakes are very complex, being three dimensional, unsteady and nonlinear. Accurate prediction of the rotor wake is one of the major challenges in rotorcraft CFD. One of the most important characteristics of the rotor wake flow is the shedding of strong tip vortices and the interaction of these vortices with the rotor blades (Blade Vortex Interaction, BVI). It is also difficult to model all of the complicated blade motions which influence the rotor wake. Therefore, the accurate capture of vortical wake flows of rotors is very important for the accurate prediction of blade loading, rotorcraft performance, vibration, and acoustics. There is still a need for higher order accurate and parallel algorithms that can capture and preserve the vortex flow over long distances at a lower computational cost.

For the rotor simulations, the flow solver PUMA2 has been modified for the solution of unsteady aerodynamics with moving boundaries, thus including both hover and forward flight conditions. The flowfield is solved directly in the inertial reference frame where the rotor blade and entire grid are in motion through still air at a specified rotational and translational speed. The computational grid is moved to conform to the instantaneous position of the moving boundary at each time step. The solution at each time step is updated with an explicit algorithm that uses the 4-stage Runge-Kutta scheme. Therefore, the grid has to be moved four times per time step and it is required to recalculate only the grid velocities and face normals for the specified grid motion at each time. [19].

Inviscid, time-accurate Euler computations for a two bladed helicopter rotor in hover have been performed using PUMA2 [1]. A rectangular, untwisted and untapered two bladed rotor model with NACA 0012 cross section and an aspect ratio of 6 was created using IDEAS. The blades have flat tips, chord of 1

m and radius of 6 m, 8 degrees collective pitch angle and 0.5 degrees pre-cone angle. The blade root cutout location is about 1 chord. A 3-D unstructured grid for the two bladed rotor geometry is generated using the GRIDGEN software. The cylindrical computational domain has nearly 1.3 million cells and 2.6 million faces. It extends to 4 radii away from the blade tips and 2 radii down and up from the rotor disk in the vertical direction. The rotational speed is chosen as 25 rad/sec to simulate an experimental hover case (Ref. [20]) of 8 degrees pitch with 1250 RPM and tip Mach number of 0.439. A 2-stage Runge-Kutta time integration method with Roe's numerical flux difference scheme with CFL number of 0.9 is used in the computations. A zero-normal-velocity boundary condition (considering also the grid velocities) is applied on the rotor blades surfaces and a Riemann boundary condition is applied at the outer boundaries of the computational domain. The computations are performed on a parallel PC cluster (COCOA-2) consisting of 40 800 MHz Pentium III processors and 20 GB RAM. This machine is extremely cost effective compared to parallel supercomputers. The rotor simulation was run on 16 processors, and the average memory consumed per processor was 110.25 MB. The computations for one revolution took nearly 15.5 days. The results of this simulation have been presented in Reference [1]. The computational steering system POSSE, which is coupled to PUMA2, has also been tested with the rotor flow simulations in Reference [1].

Iso-surfaces are very important and useful visualization tools. These are routinely used to visualize quantities such as Mach number, vorticity, and pressure. An iso-surface routine has been written in C++ and coupled to PUMA2 to give real-time iso-surfaces of pressure, Mach number, entropy, etc. These surfaces can be displayed on standard monitors or on the RAVE using OpenGL and stereographics. The body/surface grid and the iso-surface can also be colored according to some flow variable (e.g., pressure). In addition, these iso-surfaces are computed in parallel so the approach is scalable to very large grids and thousands of processors. The drawing of the iso-surfaces is performed locally on the client machine, and it does not effect the speed of the simulation being performed on the server or Beowulf cluster. Visualization of the flowfield with iso-surfaces and surface contours helps the user qualitatively interact with their simulations. It is also possible to get quantitative results in real time with routines coupled with the flow solver.

Two significant advantages arising from the use of POSSE within PUMA2 are that of *scalability* and *dimensional reduction*. The scalability comes from the fact that the extraction of iso-surfaces is done on a parallel machine in a scalable manner rather than the traditional and non-scalable way of consolidating the data into a file and post-processing it. The dimensional reduction comes from the fact that the data required for the CFD simulation lives in higher dimensional space (3-D) than the data that is required for visualization (which are in 2-D and 1-D space for iso-surfaces and chord plots, respectively). This approach is also scalable both in "space" and "time." By monitoring a time-dependent simulation, the entire time history can be accessed, whereas it would be prohibitive to store the time history in files which could possibly take tens or even hundreds of Gigabytes of disk space even for a moderately complex case.

Copyright © 2003 by ASME

While it is important to compute and display iso-surfaces rapidly, there are many other visualization tasks which are best performed in off-the-shelf graphics packages, such as Tecplot [21], Ensight [22,23], or Fieldview [24]. It is not practical to replicate the powerful features of these capable software packages in in-house codes, hence they are coupled with the computational steering system POSSE for several visualization tasks. Qualitative images are useful for debugging and steering, but often detailed quantitative images are necessary, and these are often performed best in off-the-shelf graphics software. All the iso-surfaces can be optionally filtered through a user-specified Tecplot layout file to create a powerful and highly-customized visualization display on a separate Tecplot window. For example, for the validation of the flow solvers, the surface pressure coefficient distributions at several locations need to be compared with the experimental measurements. This can be done effortlessly using the Tecplot feature of the GUI. This is also scalable since the flowfield can be processed in parallel and only a subset of the results need to be sent to the graphics workstation. The key is to use dimensional reduction and not try to post-process the entire solution on a workstation (which is often not feasible due to memory limitations).

Figures 4 shows iso-surfaces for a helicopter rotor flow using the Tecplot feature of the POSSE GUI client. There is an option "Get Grid" in the POSSE GUI client, which will download the entire grid and the updated solution file and construct a Tecplot volume grid file for the user to browse locally. Several iso-surfaces can be layered on top of each other to compare the differences between two iterations.
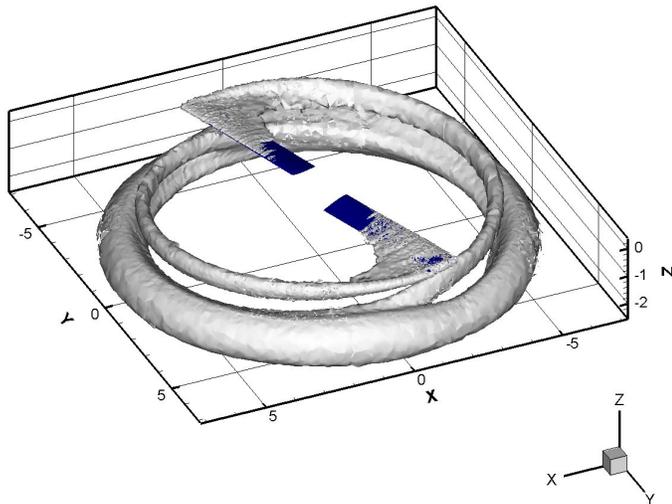

Figure 4. Entropy iso-sufaces for a CFD flow solution over the rotor blades in hover (Ref. [1]).

Ship Airwake Simulations using CFD

Shipboard launch and recovery is one of the most challenging, training intensive and dangerous of all helicopter flight operations [25]. Ship-based helicopters must operate off rolling and pitching decks, in adverse weather conditions, and often within close proximity to the superstructure of the ship. The wind over deck (WOD) conditions are a very important issue. Disturbances due to the turbulent, unsteady flowfield of the ship can have significant effect on pilot workload, and thus the fidelity of the dynamic interface simulation is dependent on

an accurate representation of the ship's airwake. A logical solution is to couple the flight dynamics simulation with time accurate CFD solutions of the ship airwake. Understanding and modeling the ship airwake presents a number of technical challenges. Complex ship geometries (with superstructures and sharp edges), atmospheric boundary layers, atmospheric turbulence, and helicopter/ship airwake interactions all add to the complexity of the flowfield. Some of the key flow features of the airwake are: unsteadiness, large regions of separated flow, vorticity, and low Mach number.

Steady-state and time-accurate inviscid CFD simulations of an LHA ship airwake have been performed using PUMA2 [25]. A 3-D unstructured grid for the full-scale LHA geometry is generated using the GRIDGEN software, which has 854,072 cells in a rectangular computational domain as shown in Figure 5. The flow case represents 0 degree yaw angle and Mach number of 0.18 (30 knot of relative wind speed). A 4-stage Runge-Kutta explicit time integration algorithm with Roe's flux difference scheme is used with CFL numbers of 2.5 and 0.8 for the steady and unsteady computations, respectively. A zero-normal-velocity boundary condition is applied on the ship surface and water surface (bottom surface of the domain) and a Riemann boundary condition is applied at all other faces of the domain. The pseudo steady-state computations are performed using local time-stepping and initialized with freestream values. The time accurate computations are started from the pseudo steady state solution, and the simulation time step (44 µseconds) is determined by the smallest cell size in the volume grid. It takes nearly 22870 iterations to simulate 1 second of real flow (~2 days on 40 processors) with the current grid on the parallel PC cluster COCOA-2. An instantaneous iso-surface of vorticity magnitude over the LHA ship and the velocity magnitude contours at several y-z planes over the LHA at the simulation time of 20 seconds are shown in Figures 6 and 7, respectively. Complex flow features such as deck-edge vortices, separated vortical regions on the deck, complex island wake can be observed in these figures. Stereographics displays are useful to visualize and better understand the details of such complex flows. The results of the steady-state and time-accurate simulations of the LHA ship airwake, which a helicopter dynamics simulation model was interfaced with, have been discussed in detail in Reference [25]. This study clearly indicates that time-varying ship airwake has a significant impact on aircraft response and pilot control activity when the helicopter is operating in or near a hover relative to the ship deck (stationkeeping).
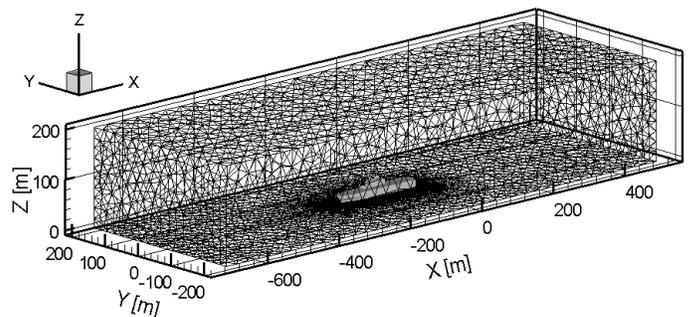

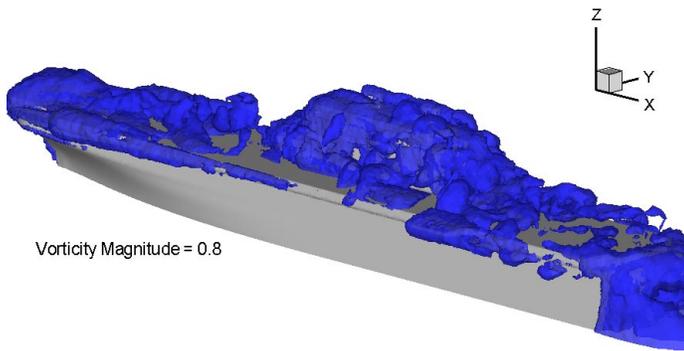**Figure 5. Unstructured grid domain around the LHA ship.**

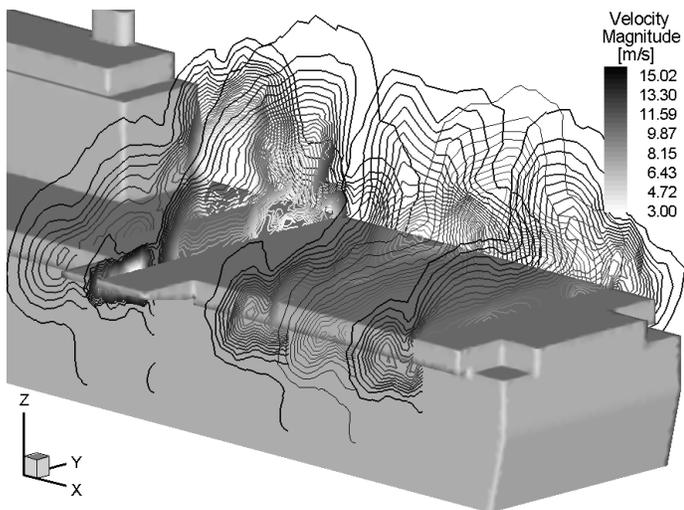**Figure 6. Vorticity magnitude iso-surface for a flow solution over the LHA ship.**

Figure 7. Velocity magnitude contours at several y-z planes over LHA

Helicopter Fan-in-Fin Flow Simulations using CFD

PUMA2 has also been modified to obtain numerical simulations of the flowfield around the RAH-66 Comanche helicopter with and without the FANTAIL operating [26, 27]. References [26] and [27] discuss the unsteady and steady solutions, respectively. The FANTAIL is modeled using two different methods: uniform momentum theory and blade element theory. The effect of the FANTAIL on the overall flowfield, the effect of blade pitch settings on the fan thrust and aerodynamic forces and moments, the unsteady thrust response, and the control effectiveness of the helicopter are analyzed for various flight regimes (hover, forward flight and sideward flight).

Inviscid, steady computations have been performed using PUMA2 on the PC cluster COCOA2. Figure 8 shows the surface pressure coefficient contours for an RAH-66 Comanche helicopter with FANTAIL operating for a case with forward speed of 85.9 knots and a fan thrust coefficient of 0.08. This solution is obtained using momentum theory and the fan thrust is given as an input instead of the blade pitch angle. The pressure coefficient contours and the velocity vectors on a 2-D slice through the FANTAIL are shown in Figures 9 and 10, respectively, for a forward flight case with 150 knot forward

speed and collective pitch setting of 20 deg. The FANTAIL is modeled using blade element theory for this case. In figure 10 it is clear that the flowfield is separated inside the fan housing creating extremely complicated flow patterns. References [26] and [27] show that these solutions agree very well with measured data. Stereographics displays are invaluable for viewing and understanding the complex, three-dimensional nature of flows such as those in the FANTAIL. Design engineers have to use these immersive systems not only to achieve specific results, but to obtain an intuitive feel for these results to enable design optimizations.
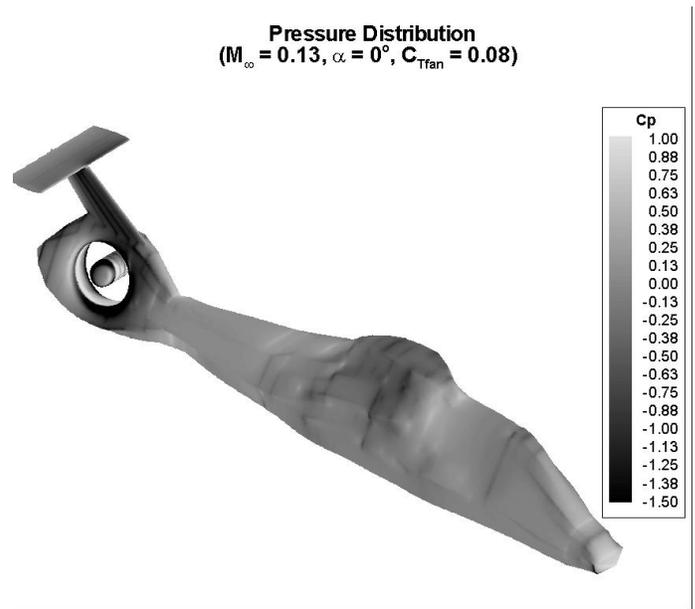

Figure 8. Surface pressure coefficient contours for RAH-66 Comanche helicopter with FANTAIL (Ref. [27]).
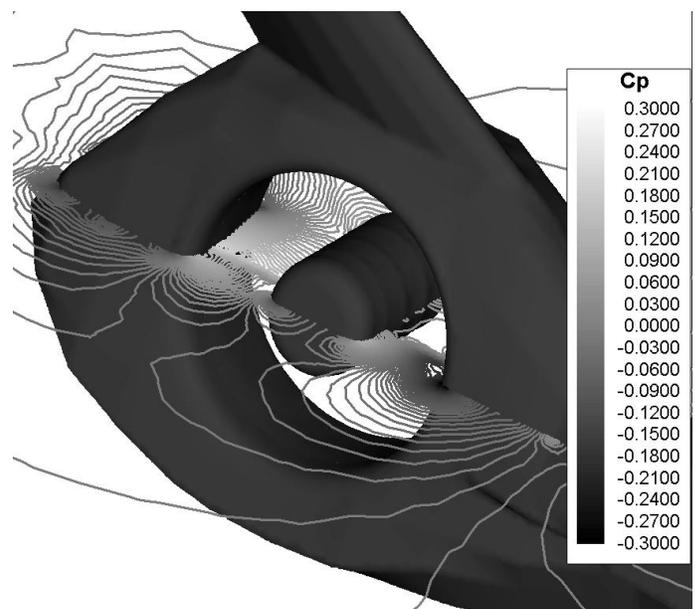

Figure 9. Pressure coefficient contours on a 2-D slice through FANTAIL (courtesy E. Alpman).
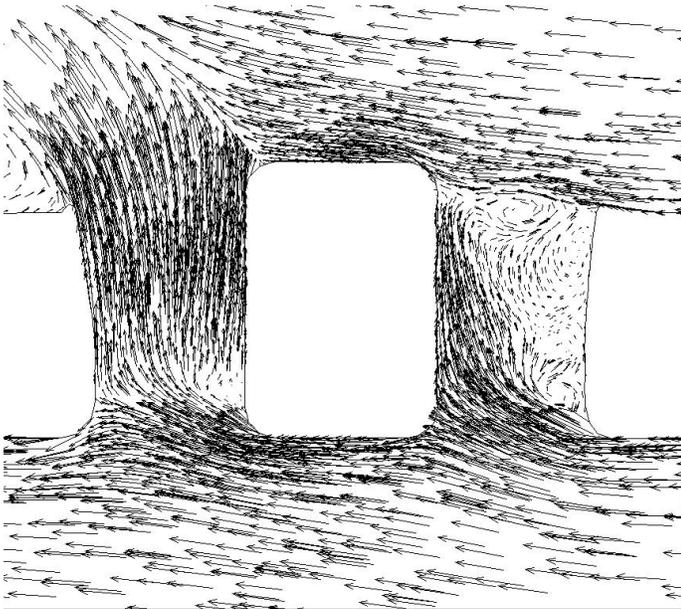
Figure 10. Velocity vectors on a 2-D slice through FANTAIL (Ref. [27]).

Figure 11. Instantaneous pressure distribution on the landing gear surface (Ref. [17]).

Landing Gear Simulations using CFD

Airframe noise (especially on approach to landing) is becoming an important noise source, as fan and jet noise is reduced. The noise due to separated flow around landing gear and flaps can be significant. The landing gear configuration is an especially challenging test case for which massive flow separation occurs. [9].

A numerical simulation study of time-accurate flow and noise due to a detailed landing gear configuration using PUMA2 is presented in References [17] and [18], where both the aerodynamic and acoustic results are discussed. PUMA2 has been combined with a module, which uses the Ffowcs Williams–Hawkings (FW-H) equation with solid and permeable integration surfaces, to compute near- and far-field noise due to landing gear. The time-accurate monotone integrated large eddy simulations (MILES) are performed on the PC cluster COCOA2. Figure 11, which is a display of the instantaneous pressure contours on the landing gear surface, shows the uneven pressure loading acting on gear components such as the wheels or support struts. Figure 12 shows a 3-D representation of some vortex filaments shed from various gear components, and highlights the impact of the upstream elements' wake onto gear elements at downstream locations. Flow separation from the fore wheel and wake impingement on the aft are expected to generate large unsteady pressure fluctuations and therefore noise. Unsteady simulations such as these are really four-dimensional (space and time) problems, where complex 3-D flowfields change in time. In the future simulations like these can be studied with stereographics combined with stereo sound.
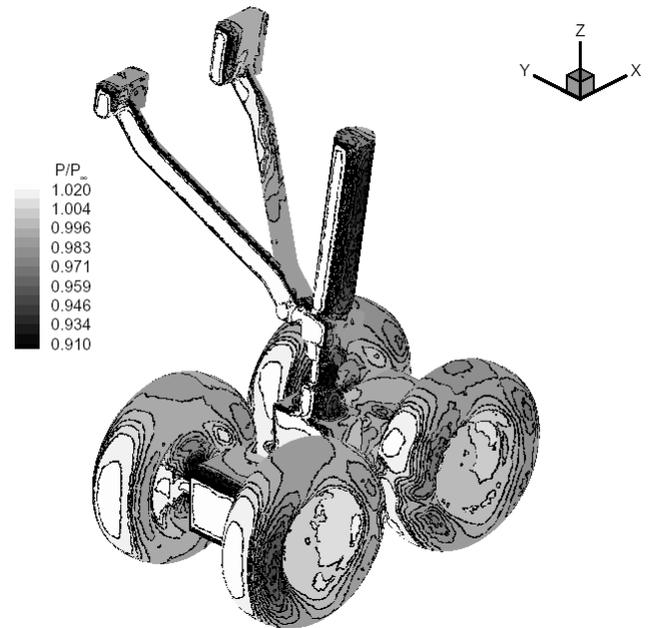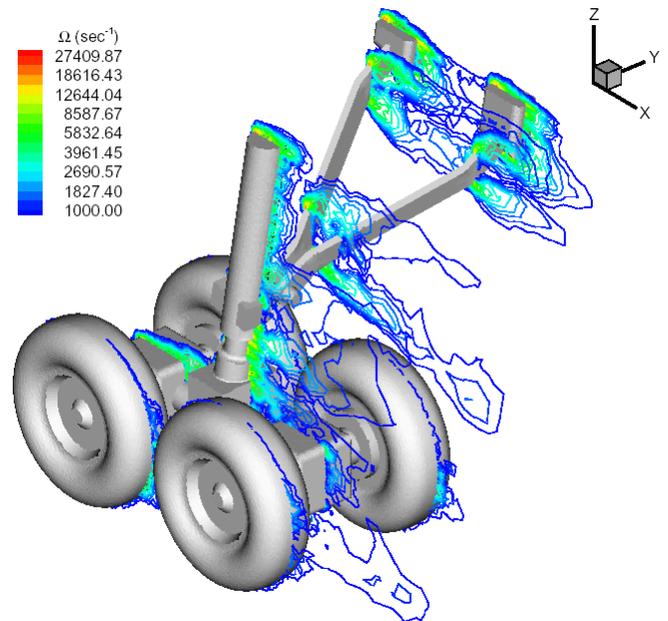

Figure 12. Instantaneous isovorticity lines for landing gear (Ref. [17]).

CONCLUSIONS

In this paper we have presented several complex fluid problems which are extremely difficult to understand using conventional 2-D graphics. Iso-surfaces in 3-D space can be used to highlight some flow features, but these really need to be viewed using stereographics. For 4-D problems that change in both space and time, it will be essential to use stereographics. It is also very important to have libraries such as POSSE to visualize large-scale parallel simulations. Large parallel simulations need scalable visualization solutions such as these. Object oriented C++ allows one to write libraries such as

POSSE which is easy to use and powerful. We have demonstrated how POSSE can be coupled to unsteady CFD codes, such as PUMA2.

ACKNOWLEDGMENTS

REFERENCES

[1] Modi, A., Sezer-Uzol, N., Long, L.N., and Plassmann, P.E., "Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations," AIAA 2002-2750, Presented at 32nd AIAA Fluid Dynamics Conference and Exhibit, St. Louis, Missouri, 24-27 June, 2002.

[2] Arsenault, L., Kelso, J., Kriz, R., and Das Neves, F.D., "DIVERSE: a Software Toolkit to Integrate Distributed Simulations with Heterogeneous Virtual Environments," CS Technical Report TR-01-18, Virginia Tech, 2001.

[3] Cruz-Neira, C., Sandin, D.J., DeFanti, T.A., Kenyon, R.V., and Hart, J.C., "The CAVE: audio visual experience automatic virtual environment," Communications of the ACM, 35, pp. 64-72, 1992.

[4] Cruz-Neira, C., "Applied Virtual Reality," Course 14 Notes, In Proceedings of SIGGRAPH '98 (Orlando FL, July 1988), ACM Press.

[5] Modi, A., "POSSE: Portable Object-oriented Scientific Steering Environment," http://posse.sourceforge.net, 2001.

[6] Modi, A., "Software System Development for Real-Time Simulations Coupled to Virtual Reality for Aerospace Applications", *PhD Thesis,* Dept. of Computer Science and Engineering, The Pennsylvania State University, August 2002.

[7] Modi, A., "COst effective COmputing Array-2", http://cocoa2.ihpca.psu.edu, 2001.

[8] Modi, A., and Long, L. N., "Unsteady Separated Flow Simulations Using a Cluster of Workstations," *AIAA 2000-0272, Presented at 38th Aerospace Sciences Meeting & Exhibit*, 10-13 January, 2000, Reno, NV.

[9] Long, L. N., and Modi, A., "Turbulent Flow and Aeroacoustic Simulations using a Cluster of Workstations," *NCSA Linux Revolution Conference*, June 2001, Illinois.

[10] Fakespace Systems, *RAVE,* http://www.fakespace.com/

[11] Gu, W., Eisenhauer, G., Kraemer, E., Schwan, K., Stasko, J., Vetter, J., and Mallavarupu, N., "Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs," *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computatio*n, pp. 433–429, February 1995.

[12] Ba, I., Malon C., Smith, B., "Design of the ALICE Memory Snooper," http://www.mcs.anl.gov/ams, 1999.

[13] Peter S. Pacheco, "Parallel programming with MPI," Morgan Kaufmann Publishers, 1997.

[14] The Fast Light ToolKit (FLTK), http://www.fltk.org, 1998.

[15] The Visualization ToolKit (VTK), http://www.kitware.com/vtk.html, 2000.

[16] Modi, A., Long, L.N., and Plassmann, P.E., "Real-Time Visualization of Vortex-Wake Simulations using Computational Steering and Beowulf Clusters," VECPAR 2002, June 2002, Lisbon, Portugal.

[17] Souliez, F.J., "Parallel Methods for the Computation of Unsteady Separated Flows Around Complex Geometries," *PhD Thesis*, Dept. of Aerospace Engineering, The Pennsylvania State University, August 2002.

[18] Souliez, F.J., Long, L. N., Morris, P. J., and Sharma, A., "Landing Gear Aerodynamics Noise Prediction using Unstructured Grids," *AIAA Paper 2002-0799, Presented at 40th AIAA Aerospace Sciences Meeting & Exhibit*, January 14-17, 2002, Reno, NV. (to appear International Journal of Aeroacoustics, 2002).

[19] Sezer-Uzol, N., "High Accuracy Wake and Vortex Simulations using a Hybrid Euler/Discrete Vortex Method," *M.S. Thesis,* Dept. of Aerospace Engineering, The Pennsylvania State University, May 2001.

[20] Caradonna, F.X., and Tung, C., "Experimental and Analytical Studies of a Model Helicopter Rotor in Hover," *Vertica* Vol. 5, No. 2, pp. 149-161, 1981.

[21] Tecplot, http://www.amtec.com/

[22] Ensight, http://www.ceintl.com/

[23] Turnage, A., "Reducing Aircraft Noise with Computer Graphics," IEEE Computer Graphics, pp 16-21, May 2002.

[24] Fieldview, http://www.ilight.com/

[25] Lee, D., Sezer-Uzol, N., Horn, J.F., and Long, L. N., "Simulation of Helicopter Shipboard Launch and Recovery with Time-Accurate Airwakes," AHS 59th Annual Forum, Phoenix, AZ, May 6-8, 2003.

[26] Alpman, E., and Long, L.N., "Unsteady RAH-66 Comanche Flowfield Simulations including Fan-in-Fin," AIAA Paper 2003- 4231, 16th AIAA Computational Fluid Dynamics Conference, Orlando, Florida, 23-26 Jun 2003.

[27] Alpman, E., Long, L.N., and Kothmann, B., "Comanche Fan-in-Fin Simulations," AHS 59th Annual Forum, Phoenix, AZ, May 6-8, 2003.