

Scalable Computational Steering for Visualization/Control of Large-Scale Fluid Dynamics Simulations

Anirudh Modi,* Nilay Sezer-Uzol,[†] Lyle N. Long,[‡] and Paul E. Plassmann[§]
Pennsylvania State University, University Park, Pennsylvania 16802

The development, integration, and testing of a general-purpose “computational steering” software library with a three-dimensional Navier–Stokes flow solver is described. For this purpose, the portable object-oriented scientific steering environment (called POSSE) library was used. This library can be coupled to any C/C++ simulation code. The paper illustrates how to integrate computational steering into a code, how to monitor the solution while it is being computed, and how to adjust the parameters of the algorithm and simulation during execution. The simulations typically run on a parallel computer, whereas the visualization is performed both on the parallel machine and on other computers through a client/server approach. In addition, the visualizations can be displayed using virtual reality (stereographics) facilities to better understand the three-dimensional nature of the flowfields. A key advantage of our interactive CFD system is its scalability. For large-scale simulations it is often not possible to postprocess the entire flowfield on a single computer due to memory and speed constraints. Therefore, scalable interactive computational steering and monitoring systems are essential. Example applications are presented including flow over a helicopter fuselage, a helicopter rotor, a ship airwake, and a landing gear. The advantages of using object-oriented programming are also discussed.

Nomenclature

| | | |
|----------------|---|--|
| \mathbf{b} | = | grid velocity |
| C_p | = | coefficient of pressure |
| C_T | = | thrust coefficient |
| c | = | chord of the rotor blade |
| e_0 | = | total energy |
| \mathbf{F} | = | inviscid fluxes with the rotational terms |
| \mathbf{F}_v | = | viscous flux terms |
| h_0 | = | total enthalpy |
| N_f | = | total number of grid faces |
| n | = | average number of grid cells in one direction |
| n_f | = | average number of grid faces in one direction |
| n^3 | = | total number of grid cells in a three-dimensional uniform grid |
| n_f^3 | = | total number of grid faces |
| m | = | total number of time steps |
| P | = | total number of processors |
| p | = | pressure |
| \mathbf{Q} | = | vector of the flowfield variables in conservative form |
| q_j | = | heat flux |
| R | = | radius of the rotor blade |
| Re | = | Reynolds number |
| Re_l | = | Reynolds number based on the integral length scale of turbulence |
| \mathbf{U} | = | absolute flow velocity |
| τ_{ij} | = | viscous stress tensor |

Introduction

PARALLEL simulations now play an important role in all areas of science and engineering (fluid dynamics, electromagnetics, structural dynamics, materials science, etc.). Fluid dynamics simulations for realistic aerospace engineering applications often require the computations of complex three-dimensional, unsteady, separated, and turbulent flows. Turbulence modeling plays a vital role in the simulations of these complex flows. Various numerical methods have been proposed in the literature for turbulent flow calculations: direct numerical simulations (DNS), large-eddy simulations (LES), Reynolds-averaged Navier–Stokes simulations, and the hybrid methods, for example, detached eddy simulations (DES), etc., that couple these simulations. 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 because 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.

Whereas a tremendous amount of work has gone into developing parallel computational fluid dynamics (CFD) software, little has been done until the past few years to develop parallel computational steering tools that can be integrated with such parallel 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 postprocessing of data. However, for large-scale, parallel aerospace CFD simulations, it is often not possible to postprocess the entire flowfield on a single computer due to memory and speed constraints. Large parallel supercomputers can store thousands of times more data than a typical workstation. Also, the interaction with the large-scale data, which is especially important at the early stages of the computations to tune the several flow and input parameters, becomes very difficult and time consuming for such large-scale simulations. Therefore, interactive parallel (and scalable) computational steering and monitoring systems are necessary to perform efficiently large-scale, parallel aerospace CFD simulations. The system described here allows the steering, visualization, and CFD to all be run in a scalable manner. For example, the majority of the work

Received 19 January 2004; revision received 4 June 2004; accepted for publication 4 June 2004. Copyright © 2004 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. Copies of this paper may be made for personal or internal use, on condition that the copier pay the \$10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 0021-8669/05 \$10.00 in correspondence with the CCC.

*Ph.D. Candidate, Department of Computer Science and Engineering; currently Senior Software Engineer, Intel Corporation, Portland, OR 97229; anirudh@anirudh.net.

[†]Ph.D. Candidate, Department of Aerospace Engineering; nilay@psu.edu.

[‡]Professor, Department of Aerospace Engineering; ln1@psu.edu. Associate Fellow AIAA.

[§]Associate Professor, Department of Computer Science and Engineering; plassmann@cse.psu.edu.

required for the visualization is done in parallel with essentially no communication required. In addition, the computational steering can be a means to make simulations appear more like experiments. That is, one can probe the simulation in arbitrary ways, in real time, in a manner similar to that of an experimentalist. This represents a shift from the old postprocessing way of doing large-scale simulations that will make simulations infinitely more useful.

The number of grid cells and the resulting computational cost required to simulate turbulent flow increase with the Reynolds number at a rate depending on the numerical method used. For example, DNS, which resolves all scales of turbulence, is computationally very expensive because the smallest length scale of the computational grid is required to be on the order of the size of the finest turbulent eddies. If we let n be the number of grid cells along each dimension of a turbulent flow simulation using a uniform grid, the total number of grid cells required for the three-dimensional simulation will be of order n^3 . For DNS, the number of grid cells required has been estimated¹ to scale with Reynolds number as $4.4 \cdot Re^{2.25}$, and the total number of time steps m scales¹ as roughly $23 \cdot n$. For a Reynolds number of 10^7 , this estimate indicates that approximately 10^{15} grid cells would be required by a DNS simulation. For comparison, the largest case possible today is about 6.9×10^{10} (4096³ grid points on the Earth simulator computing system²).

In LES, the three-dimensional, time-dependent motion of large scales is computed explicitly, and small-scale effects are modeled. As a result of the modeling of small-scales, LES does not require as much resolution as DNS, and, therefore, the computational cost is less than DNS. For LES with no wall function, the number of grid cells required n^3 is estimated¹ to scale with Reynolds number as $Re^{1.8}$. With the use of wall functions or other numerical methods such as DES, significantly fewer grid cells would be required.

We can use the preceding scaling relations to estimate the largest Reynolds number able to be modeled by DNS and LES on a moderately sized parallel computer as a function of year. We employ a commonly used derivative of Moore's law, that the computational speed of computers will double every 18 months. Based on these assumptions, Fig. 1a shows the largest Reynolds number that can be solved with the computing power available with a moderately sized parallel computer using DNS and LES for the same values of n^3 and m from the present until the year 2050. Our parallel computer model assumes that the problem is solved on a dedicated machine with 100 processors running for 10 days. We assume that the processor speed doubles every 1.5 years starting from a 4-GHz processor in 2004. For this processor, we assume a sustained performance of 800 million floating-point operations can be obtained (based on two operations/clock cycle and achieving 10% of peak speed). The total number of operations required to solve the problem is proportional to $m \cdot n^3$, and we assume that 1000 operations per grid cell per time step are required. For the results shown, we assume that the number of time steps m required for LES depends on n based on the same relation as for DNS. Given the number of operations available to our simulation as a function of year from our parallel computer model, we can estimate the largest Reynolds number that can be solved by DNS and LES. These estimates are shown in Fig. 1a.

Note that the number of grid cells n^3 will be the same for DNS and LES based on our assumption that the number of time steps required m is the same for both methods. In Fig. 1b, we show the number of grid cells required as a function of year for the largest Reynolds number problems shown in Fig. 1a. The total memory required for the simulation is proportional to the number of grid cells. If we assume that approximately $400 \cdot n^3$ bytes of memory (assuming 50 double-precision variables) per grid cell is required for the simulations, then the total memory required for these simulations exceeds that available from a single work-station. However, as long as enough memory is available, the required floating-point operations will be the limiting factor in the preceding calculations. Specifically, for the problem we assume that we can solve in 2004, n^3 is 2.3×10^9 . This number would require approximately 10 GB of memory per processor on the parallel computer for storing the grid

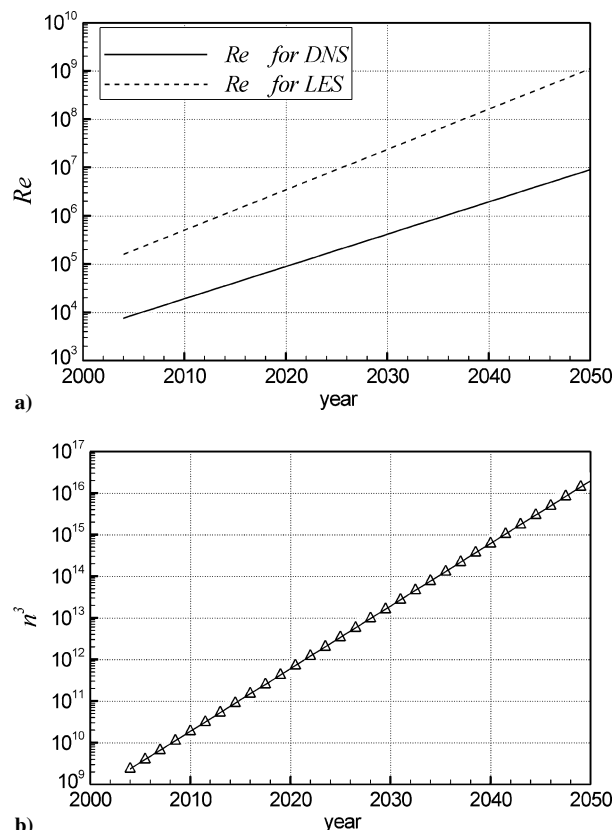


Fig. 1 Estimates of a) the largest Reynolds number and b) number of grid cells n^3 that can be solved with available computer power using DNS and LES as a function of year.

cell information. This memory requirement seems large. However, note that because the overall computational requirements grow at a higher order (with respect to n) than the memory requirements, we can assume that the total memory available to solve future problems will be sufficient. Spalart³ predicted that it will be possible in 2080 to perform DNS simulation for a chord Reynolds number of 7×10^7 for a wing, which is consistent with our analysis. Therefore, LES, Reynolds stress methods, or hybrid methods will be essential to perform realistic aerospace simulations, both now and in the future, and the interactive and scalable computational steering approach has to be adopted for large-scale computations such as these turbulent flow simulations.

An easy to use, general-purpose computational steering library: portable object-oriented scientific steering environment (POSSE),⁴⁻⁶ has been developed that can be easily coupled to any existing C/C++ simulation code. POSSE can be downloaded from SourceForge.⁴

Computational Monitoring and Steering Library

The computational monitoring and steering library POSSE⁴ is written in C++, by the use of 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)⁷ library, to obtain an interactive CFD system. This system is being successfully used to monitor and steer several large flow simulations over helicopter and ship geometries, thus providing the user with a fast and simple debugging and analysis mechanism, where the flow and convergence parameters can be changed dynamically without having to stop or restart the simulation. This CFD system, which primarily runs on an in-house Beowulf

cluster, the cost-effective computing array-2 (COCOA-2),^{8–10} has been coupled to our virtual reality (VR) system, Fakespace RAVE,¹¹ to obtain near real-time visualization of the three-dimensional solution data in stereographic mode. This ability to become 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 flow solver 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 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, for example, on a specialized graphics workstation for visualization.

For CFD simulations, it is important for the surface contours of flow variables to be computed instantaneously and sent to the visualization client 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 need 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 that 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 these 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 online, so that the application need not be stopped and restarted.

A significant amount of work has been done on computational steering systems over the past few years. Some of the well-known steering systems are Falcon¹² from the Georgia Institute of Technology, SCIRun¹³ from the Scientific Computing and Imaging Research Group at University of Utah, ALICE Memory Snooper¹⁴ from the Argonne National Laboratory, the visualization and application steering environment (VASE)¹⁵ from the University of Illinois, CUMULVS¹⁶ from Oak Ridge National Laboratory, the computational steering environment (CSE)¹⁷ from the Center for Mathematics and Computer Science in Amsterdam, pV3¹⁸ from the Massachusetts Institute of Technology, Virtue¹⁹ from the University of Illinois at Urbana-Champaign, and COVISE^{20,21} from the University of Stuttgart. A summary of the characteristics of these systems is listed in Table 1. These visualization and steering systems provide visualization techniques for displaying the application's input and output data (model exploration). VASE and SCIRun also allow algorithm experimentation by modifying a running program's algorithm and structure. Falcon and Virtue provide only performance optimization for a running application program by changing computational resources. Some of them have a suitable graphical user interface (GUI) for steering and data visualization, and most are based on a client/server model for data extraction and parameter access. The application source code has to be modified manually with program statements (at user-defined breakpoints) by the application developer for all systems except SCIRun. More detailed comparison and description of many steering systems can be found in Refs. 22–25.

Computational steering has also been attempted by coupling of a two-dimensional solver with the VR environment by Cruz-Neira et al.²⁶ (see Ref. 27) at the Electronic Visualization Laboratory

Table 1 Summary of existing steering and visualization systems

| System | Characteristics |
|---|--|
| VISUAL3 (1991) | Visualization package for three-dimensional, unstructured grid, steady or unsteady data |
| VASE (1993) | Visualization through existing packages Steering through textual input Hybrid control flow and data flow Algorithm experiments |
| Parallel Visual 3 (1994) | Parallel version of VISUAL3 For single parallel PVM simulation |
| SCIRun (1995) | Visualization through visualization modules Steering through TCL/TK user interfaces and widgets Data flow with feed back loops Algorithm experiments |
| Falcon (1995) | Two-dimensional visualization Steering user interface Only for performance optimization For thread-based parallel programs, and programs for PVM. |
| CSE (1996) | Graphical editor for multidimensional datasets Steering GUI Client/server |
| Collaborative visualization and simulation environment (1993) | Interactive collaborative postprocessing |
| COVISE Virtual Environment (1997) | Visualization user interface such as AVS or SCIRun COVER: A virtual reality rendering module, IRIS performer |
| CUMULVS (1997) | Interactive visualization through AVS Computational steering through textual input and custom TCL/TK GUI Client/server For single parallel, C, or FORTRAN, PVM simulation Performance optimization |
| ALICE (1999) | Steering GUI MATLAB® interface |
| Virtue (1999) | Interactive VR visualization Steering Only for performance optimization |
| Portable object-oriented scientific steering environment (2002) | Steering and visualization GUI through cross-platform FLTK, API, and VTK, coupled with Tecplot Client/server For single C/C++, MPI simulation |

(EVL), University of Illinois at Chicago, as early as 1993. They worked on very simple simulations of Rayleigh–Taylor instability and gravitational wave components predicted by Einstein's theory of general relativity. The datasets were relatively small and calculations were simplified by approximating theoretical algorithms and decreasing rendering quality. The steering was implemented by providing a menu to change input parameters of the simulation and then restarting the simulation when a change was detected. The computation was simultaneously performed on a separate machine that was networked with a cave automatic virtual environment (CAVE).

Although all of these software packages are powerful, the major drawback of these systems is that they are often too complex, are not object-oriented, and have a steep learning curve. It is not easy to couple these systems with existing scientific codes. It may take a significant amount of time to be productive with these systems, especially for noncomputer scientists. To address these problems, a new lightweight and portable computational steering system based on a client/server programming model has been developed.

The steering software, POSSE, is very general in nature and is based on a simple client/server model. It uses an approach similar to Falcon¹² (an online monitoring and steering toolkit) and ALICE Memory Snooper¹⁴ (an application programming interface designed

to help in writing computational steering, monitoring and debugging tools). 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. 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 application, displays the 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 are done via UNIX sockets, and threading is done using portable operating system inter-

face standard (POSIX) 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. Figure 2 shows a schematic of how POSSE can be used, and Fig. 3 shows the inheritance diagram of the POSSE classes. As seen in Fig. 2, an ongoing 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 GUI client that provides a simple user interface to all registered simulation data. The visualization code can be used to monitor interactively a dataset at various time intervals, and the GUI code can be used to steer the simulation by changing certain parameters associated with it.

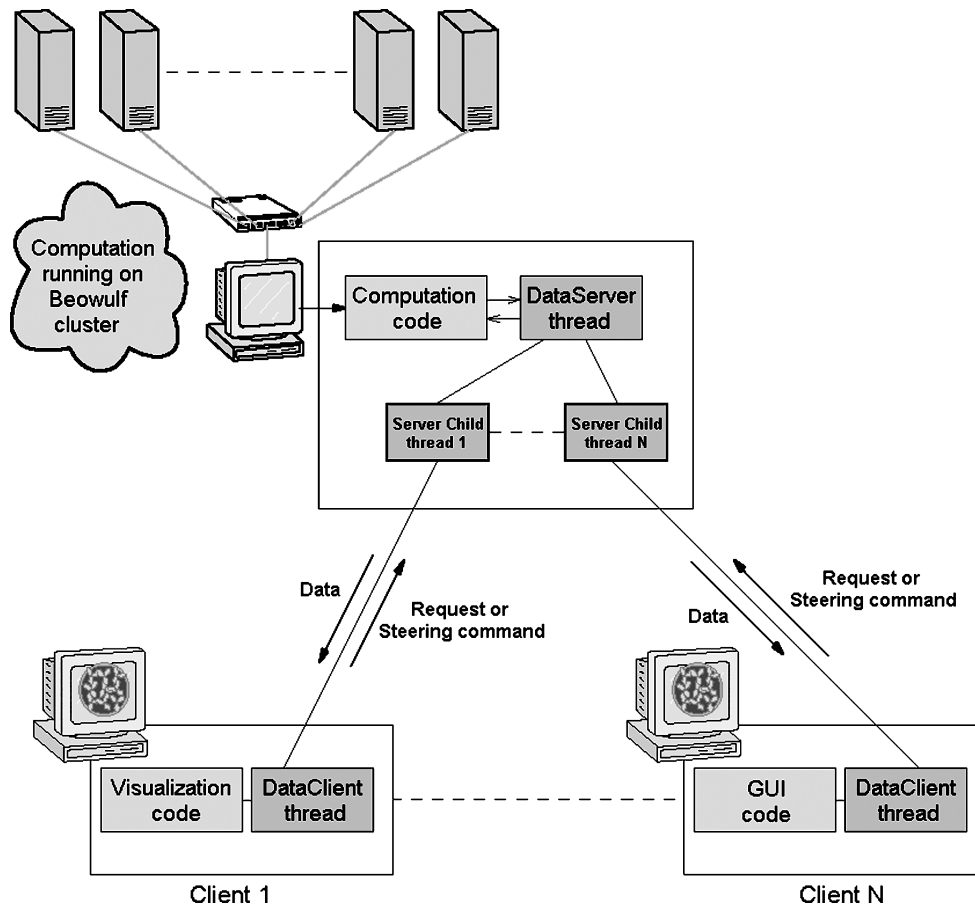


Fig. 2 Schematic of POSSE.

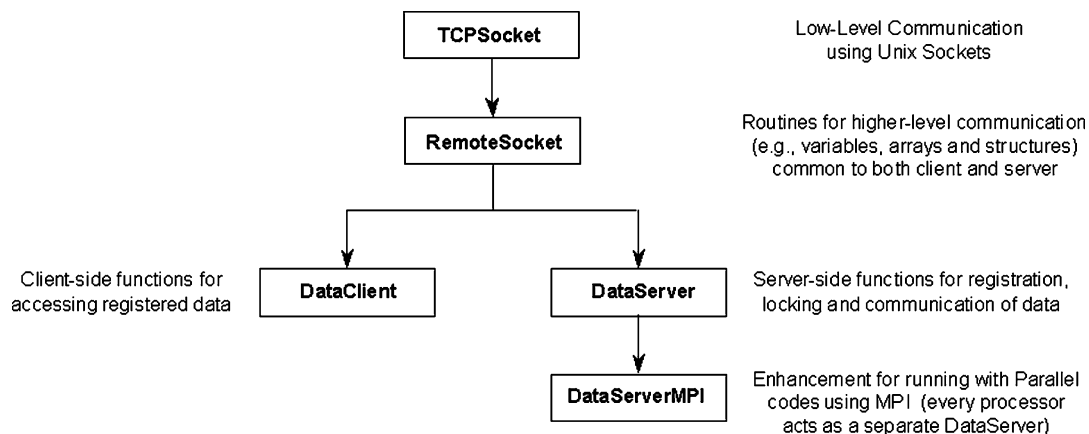


Fig. 3 Inheritance diagram for POSSE.

POSSE runs on all Win32 and POSIX compliant UNIX platforms. 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 multithreaded, supporting several clients simultaneously. It can also be easily incorporated into parallel simulations based on the MPI⁷ 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. Because of its efficient design, POSSE has low computational overhead (averaging less than 1% relative to the computation thread), making it lightweight.

Figures 4 and 5 show a simple, yet complete, POSSE client/server program in C++. As seen in Figs. 4 and 5, registered data on

```
//-----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;
}
//-----
```

Fig. 4 Simple, complete POSSE client application written in C++.

```
//-----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;
}
//-----
```

Fig. 5 Simple, complete POSSE server application written in C++.

the steering server (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 calls the 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 to use POSSE effectively. Among other applications, POSSE has been successfully used to visualize a wake–vortex simulation of several aircraft in real time.^{28,29}

Software for VR Systems

The software architecture that is considered in this paper (shown in Fig. 2) consists of two software components. The CFD simulation (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 manner (such as a steering client that changes the state of the simulation as it is executing). The POSSE software can support both the data server and any number of clients.

POSSE 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 marketed by FakeSpace,¹¹ but originally developed at the EVL. The software library to support these devices is called CAVELibs and is commercially supported by VRCO Inc.

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 DIVERSE is a collection of software modules that can be used to implement visualization clients, as an API, or to integrate heterogeneous VR environments.³⁰ In addition, VRjuggler is a software library that has been developed to support a wide variety of VE devices and has been ported to a variety of architectures.³¹ VEs are also now more commonly being supported by commercial visualization systems. For example, Ensight³² 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.

Three-Dimensional Finite Volume CFD Solver

PUMA2 is the modified version of the CFD flow solver PUMA³³ that uses the finite volume formulation of the Navier–Stokes equations for three-dimensional, internal and external, nonreacting, compressible, and unsteady- or steady-state solutions of problems for complex geometries.

The integral form of the Navier–Stokes equations with moving grids³⁴ is

$$\frac{\partial}{\partial t} \int_V Q \, dV + \int_S (\mathbf{F} \cdot \mathbf{n}) \, dS - \int_S (\mathbf{F}_v \cdot \mathbf{n}) \, dS = 0$$

$$\mathbf{F} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3, \quad \mathbf{F}_v = \mathbf{F}_{v1} + \mathbf{F}_{v2} + \mathbf{F}_{v3}$$

$$Q = \begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho e_0 \end{Bmatrix}, \quad \mathbf{F}_j = \begin{Bmatrix} \rho(u_j - b_j) \\ \rho u_1(u_j - b_j) + p\delta_{1j} \\ \rho u_2(u_j - b_j) + p\delta_{2j} \\ \rho u_3(u_j - b_j) + p\delta_{3j} \\ \rho h_0(u_j - b_j) + pb_j \end{Bmatrix}$$

$$\mathbf{F}_{vj} = \begin{Bmatrix} 0 \\ \tau_{1j} \\ \tau_{2j} \\ \tau_{3j} \\ (u_1\tau_{1j} + u_2\tau_{2j} + u_3\tau_{3j}) - q_j \end{Bmatrix} \quad j = 1, 2, 3 \quad (1)$$

where

$$p = (\gamma - 1)\rho e, \quad e_0 = e + \frac{1}{2}\mathbf{u} \cdot \mathbf{u}, \quad h_0 = e_0 + p/\rho \quad (2)$$

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 overrelaxation (SOR) schemes are also implemented. It is written in ANSI C using the MPI library for message passing, and so it can be run on parallel computers and clusters. It is also compatible with C++ compilers. PUMA2 can be run to preserve time accuracy or as a pseudounsteady formulation to enhance convergence to steady state. It uses dynamic memory allocation; thus, the problem size is limited only by the amount of memory available on the machine.

PUMA2 solves steady/unsteady Euler/Navier–Stokes equations on unstructured stationary or moving grids. For the rotor simulations, the code 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 at a specified rotational speed through any freestream. 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 four-stage Runge–Kutta scheme. Therefore, the grid has to be moved four times per time step and only the grid velocities and face normals are required to be recalculated for the specified grid motion at each time step. LES can also be performed with PUMA2 (Ref. 35). Research groups have been refining and developing this code since 1997.^{35–44}

Modifications to PUMA2

To achieve the interactivity with POSSE, several modifications were made to the PUMA2 code. The POSSE server component, DataServerMPI, was added to the main() function of PUMA2. This was done by registering the cell-centered flow vector $[\rho, \mathbf{u}, v, w, p]$ and various important flow parameters in the code. Several new global variables were added to receive isosurface requests and store resulting isosurfaces. An isosurface extraction routine also had to be added to PUMA2. Because of the use of unstructured mesh data consisting of tetrahedra, a variation of the classic marching cubes algorithm⁴⁵ is used for isosurface extraction. The implementation is closely related to the marching cube algorithm except that the fundamental sampling structure here is a tetrahedron⁴⁶ instead of a cube. Because this implementation uses the flow data at the nodes of every tetrahedron, a subroutine to interpolate the flow data from cell centers to the nodes was also added to PUMA2.

Isosurfaces are important and useful visualization primitives. This tool is routinely used to visualize quantities such as Mach number, vorticity, and pressure. An isosurface routine has been written in C++ and coupled to PUMA2 to give real-time isosurfaces 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 isosurface can also be colored according to some flow variable, for example, pressure. In addition, these isosurfaces are computed in parallel so that the approach is scalable to literally billions of grid points. The drawing of the isosurfaces 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.

To compute an isosurface representation, the following steps are performed:

- 1) Cell-centered flow variables are extrapolated to tetrahedral corner points.
- 2) For each edge of each tetrahedron, if the value of the isosurface is between the value on the two nodes, then the linear interpolation of the nodes is performed to obtain the location of the isosurface.
- 3) If the surface cuts through the tetrahedron, then a triangle is obtained (two triangles in some cases).

This approach is scalable to very large grids and thousands of processors. Visualization of the flowfield with isosurfaces and surface contours helps users to interact qualitatively with their simulations. It is also possible to get quantitative results in real time with routines coupled with the flow solver.

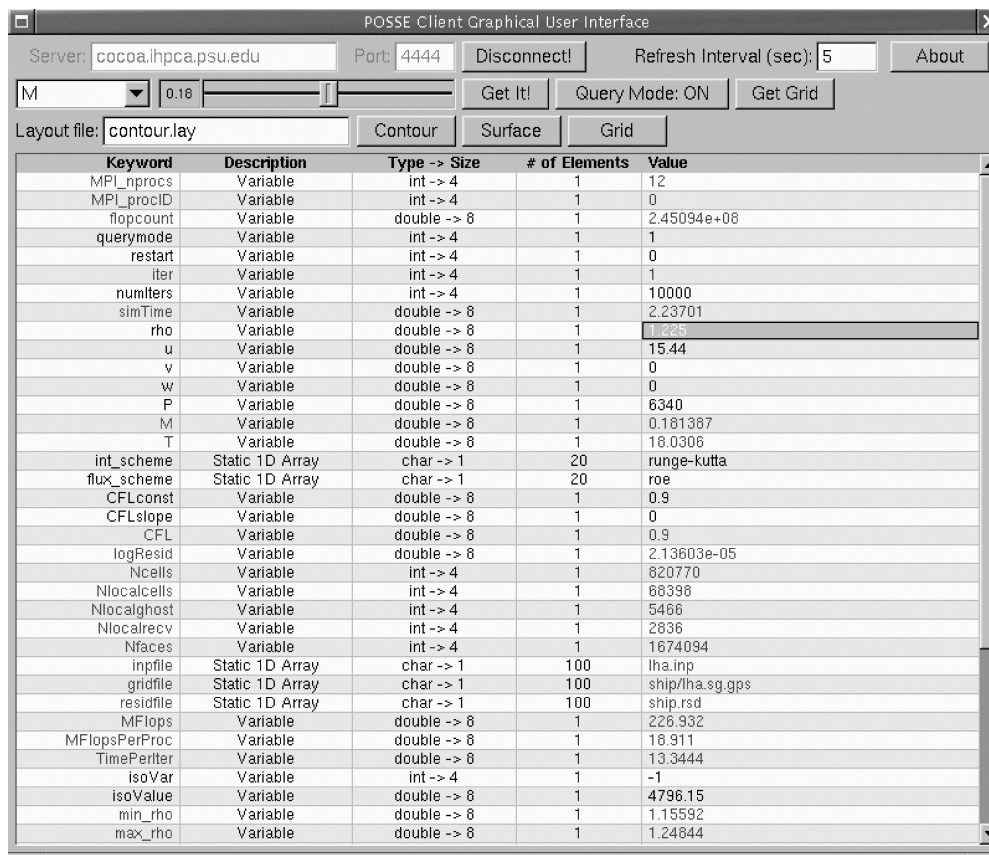
GUI

A GUI client is used to connect to the computational steering server. Figure 6 shows a screenshot from the POSSE GUI client application showing PUMA2 registered data. The selected simulation data has to be registered, and the desired actions that will take place after a change in each of these data have to be defined in the flow solver by the user before the simulation is started. The registered data can be selected, for example, as the freestream flow variables (velocity components, pressure, density, etc.), the parameters for numerical flux and integration schemes, Courant–Freidrichs–Lewy (CFL) number, relaxation parameters, etc., as seen in Fig. 6. Other solver parameters can also be registered, such as input and output file names, convergence parameters, maximum number of iterations, and a parameter used for writing the flow solution into a file at every specified number of iterations.

With the POSSE GUI client, any of the registered data can be changed to steer the simulation, and/or can be used only for monitoring purposes, for example, observing the residual for the convergence of the solution for steady-state computations. The time

required to obtain a converged steady-state solution is usually unknown at the beginning of the simulation. The same can be said about how often the updated solution needs to be written out for time-accurate computations. The ability to control and change the necessary parameters in such cases in the ongoing simulations, without stopping and restarting the simulations, is found to be very helpful. The use of POSSE GUI client for steering and monitoring the desired solver and flow parameters allows the user to save a great amount of time and effort in the initial stages of simulations in terms of debugging and experimenting with the input parameters and also enables access to the solution at any iteration in realtime.

The PUMA2 client is also used to extract and visualize isosurfaces. The client is written in C++ with the cross-platform FLTK⁴⁷ API for the GUI and the Visualization ToolKit (VTK)⁴⁸ for the three-dimensional visualization. Because 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.¹¹ Figure 7 shows a screenshot of the VTK output on a separate window. A drop-down menu is provided to choose the flow variable for which isosurfaces are requested. After the numerical value for the isosurface has been selected, a request is sent to the flow solver, which responds by extracting the isosurface for the given flow parameters on each of the processors, then collecting the isosurfaces on the master processor and sending the final isosurface to the client. There are two modes provided for querying isosurfaces. In the default mode, the user queries for isosurfaces while the flow-solver is computing the solution for the next iteration. This mode can be slow if the user wants to query several isosurfaces one after another because the flow solver cannot answer the client requests until the current iteration is over. For greater responsiveness, the user can enable the “query” mode, which temporarily halts the PUMA2 computations, so that the flow-solver can exclusively devote all of its CPU cycles to answering the client isosurface requests



The screenshot shows the POSSE Client Graphical User Interface. At the top, there are fields for 'Server: cocoa.lhpc.psu.edu' and 'Port: 4444', along with buttons for 'Disconnect!', 'Refresh Interval (sec): 5', and 'About'. Below these are controls for a variable 'M' set to 0.18, and buttons for 'Get It!', 'Query Mode: ON', and 'Get Grid'. A 'Layout file: contour.lay' is also shown. The main part of the interface is a table with columns: Keyword, Description, Type -> Size, # of Elements, and Value. The table lists various simulation parameters and their current values.

| Keyword | Description | Type -> Size | # of Elements | Value |
|---------------|-----------------|--------------|---------------|-----------------|
| MPL_nprocs | Variable | int -> 4 | 1 | 12 |
| MPL_procID | Variable | int -> 4 | 1 | 0 |
| flopcount | Variable | double -> 8 | 1 | 2.45094e+08 |
| querymode | Variable | int -> 4 | 1 | 1 |
| restart | Variable | int -> 4 | 1 | 0 |
| iter | Variable | int -> 4 | 1 | 1 |
| numiters | Variable | int -> 4 | 1 | 10000 |
| simTime | Variable | double -> 8 | 1 | 2.23701 |
| rho | Variable | double -> 8 | 1 | 1.225 |
| u | Variable | double -> 8 | 1 | 15.44 |
| v | Variable | double -> 8 | 1 | 0 |
| w | Variable | double -> 8 | 1 | 0 |
| P | Variable | double -> 8 | 1 | 6340 |
| M | Variable | double -> 8 | 1 | 0.181387 |
| T | Variable | double -> 8 | 1 | 18.0306 |
| int_scheme | Static 1D Array | char -> 1 | 20 | runge-kutta |
| flux_scheme | Static 1D Array | char -> 1 | 20 | roe |
| CFLconst | Variable | double -> 8 | 1 | 0.9 |
| CFLslope | Variable | double -> 8 | 1 | 0 |
| CFL | Variable | double -> 8 | 1 | 0.9 |
| logResid | Variable | double -> 8 | 1 | 2.13603e-05 |
| Ncells | Variable | int -> 4 | 1 | 820770 |
| Nlocalcells | Variable | int -> 4 | 1 | 68398 |
| Nlocalghost | Variable | int -> 4 | 1 | 5466 |
| Nlocalrecv | Variable | int -> 4 | 1 | 2836 |
| Nfaces | Variable | int -> 4 | 1 | 1674094 |
| inpfle | Static 1D Array | char -> 1 | 100 | lha.inp |
| gridfile | Static 1D Array | char -> 1 | 100 | ship/lha.sg.gps |
| residfile | Static 1D Array | char -> 1 | 100 | ship.rsd |
| MFlops | Variable | double -> 8 | 1 | 226.932 |
| MFlopsPerProc | Variable | double -> 8 | 1 | 18.911 |
| TimePerIter | Variable | double -> 8 | 1 | 13.3444 |
| isoVar | Variable | int -> 4 | 1 | -1 |
| isoValue | Variable | double -> 8 | 1 | 4796.15 |
| min_rho | Variable | double -> 8 | 1 | 1.15592 |
| max_rho | Variable | double -> 8 | 1 | 1.24844 |

Fig. 6 POSSE GUI to connect to the flow solver.

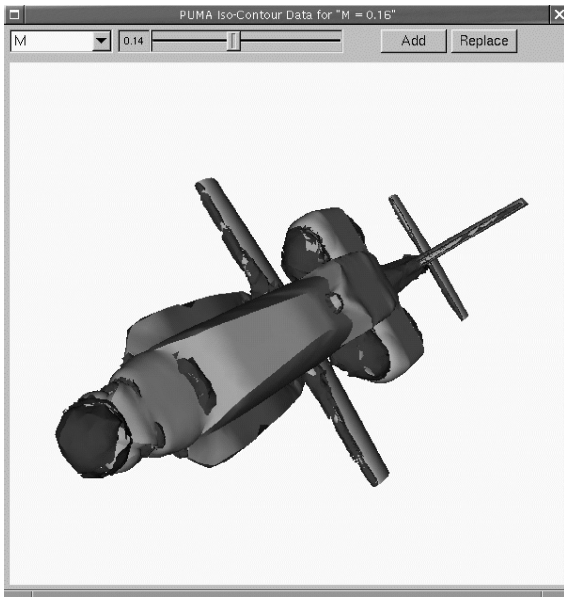


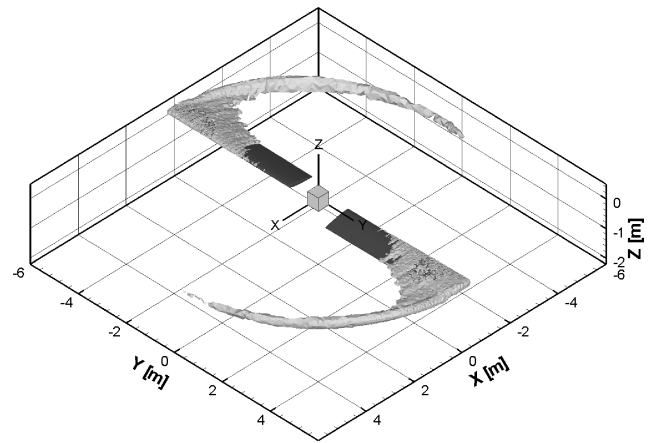
Fig. 7 POSSE client application depicting isosurfaces for a flow solution over the Apache helicopter.

without any lag. Whereas several isosurfaces can be layered on top of each other to compare the differences between two iterations by using the default mode, different isosurfaces can be investigated at a single time step in the query mode. There is also an option “get grid” that will download the entire grid and the updated solution file and construct a Tecplot volume grid file for the user to browse locally. Figures 7, 8a, and 8b show isosurfaces for a helicopter fuselage flow,³⁵ a helicopter rotor flow,^{5,40} and a ship airwake flow,^{49,50} respectively.

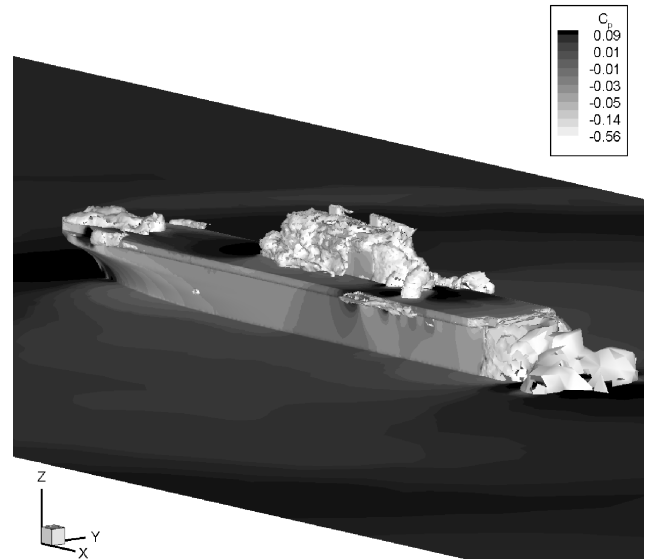
Scalability and Dimensional Reduction

Two significant advantages arising from the use of POSSE within PUMA2 are that of scalability and dimensional reduction. For an evenly distributed grid, the number of grid faces on each processor of a parallel computation is N_f/P . Here, the scalability comes from the extraction of isosurfaces being performed on a parallel machine in a scalable manner rather than the traditional and nonscalable way of consolidating the data into a file and postprocessing it. Thus, the computational time for extraction of an isosurface is $\mathcal{O}(N_f/P)$ as compared to the sequential algorithm that takes $\mathcal{O}(N_f)$ for the same procedure. The dimensional reduction stems from the data required for the CFD simulation living in higher dimensional space (three-dimensional) than the data that are required for visualization (which are in two-dimensional and one-dimensional space for isosurfaces and sectional flow variable distributions, chord plots, respectively). The total number of grid faces, N_f , is $\mathcal{O}(n_f^3)$. On the other hand, the number of triangles in an isosurface obtained from this grid is only $\mathcal{O}(n_f^2)$, which is significantly less data by a factor of $\mathcal{O}(n_f)$. Scalability and dimensional reduction combine to give an expected $\mathcal{O}(n_f^2/P)$ data coming from each processor during the parallel computation of an isosurface.

This approach is also scalable both in space and time. When a time-dependent simulation is monitored, the entire time history can be accessed, whereas it would be prohibitive to store the time history in files that could possibly take tens or even hundreds of gigabytes of disk space even for a moderately complex case. Figure 9 shows the relative size of the isosurfaces with the total size of the grid for varying X coordinate, Mach number, and C_p values for the flow over an Apache helicopter geometry. Note that the average number of triangles in an isosurface is less than 1% of the total number of grid faces for this case. Figure 10 depicts another case where two grids with vastly varying sizes are used for extraction of isosurfaces with varying values of X coordinate. Here, although the larger grid



a)



b)

Fig. 8 Flow solution over the landing helicopter amphibious (LHA) ship geometry: a) entropy isosurfaces for a flow solution over the rotor blades (grid 2) and b) entropy isosurfaces and C_p surface contours.

(with 2.3 million faces) is more than twice as large as the smaller grid (with 1.1 million faces), the average number of triangles in its isosurfaces is only about 50% more. The theoretical average difference is expected to be $(2.3/1.1)^{2/3} = 1.6352$, or 63.52% more, which is not far from the result just obtained.

Coupling of the System to Off-the-Shelf Visualization Software

Although it is important to compute and display isosurfaces rapidly, there are many other visualization tasks that are best performed in off-the-shelf graphics packages, such as Tecplot,⁵¹ Ensign,^{32,52} or Fieldview.⁵³ It is not practical to replicate the powerful features of these capable software packages in in-house codes; hence, they are used for several visualization tasks in this work. 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 of the isosurfaces can be optionally filtered through a user-specified Tecplot layout file to create a 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 efficiently done using the Tecplot feature of the

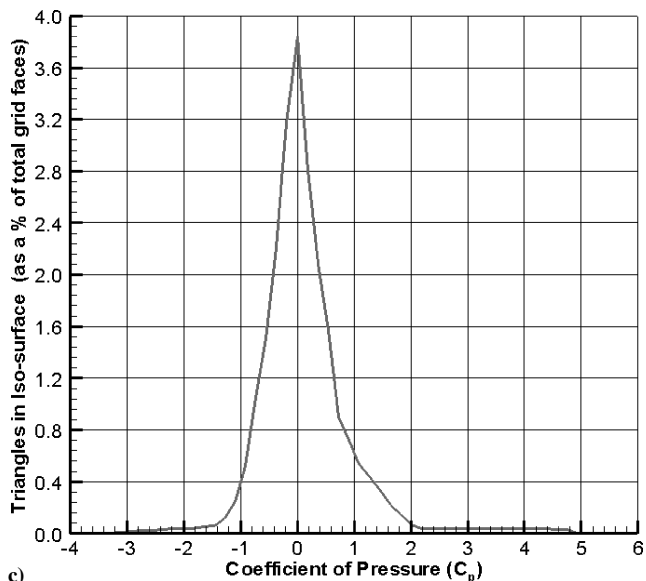
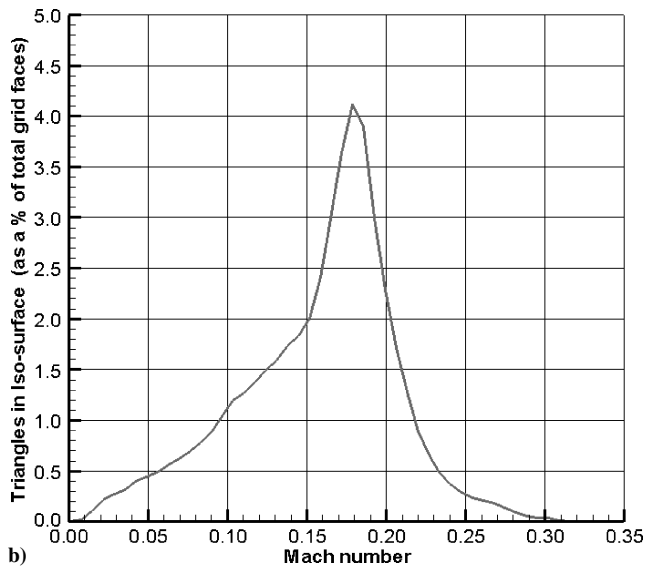
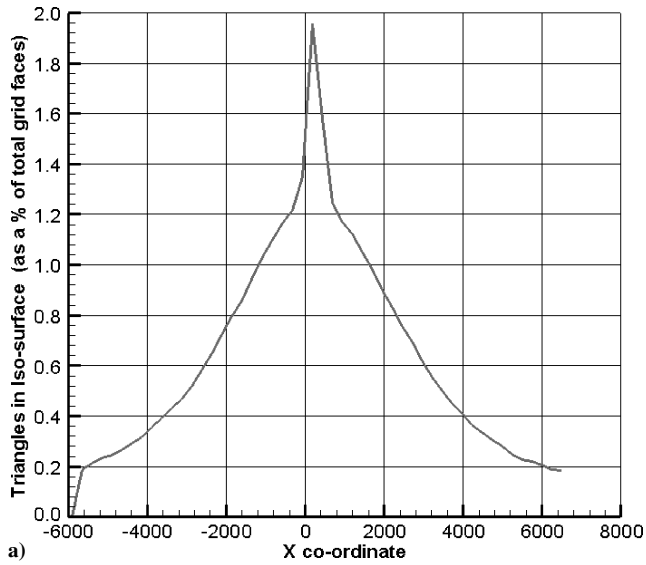


Fig. 9 Percentage of a) X coordinate, b) Mach number, and c) coefficient of pressure isosurface triangles for the Apache case.

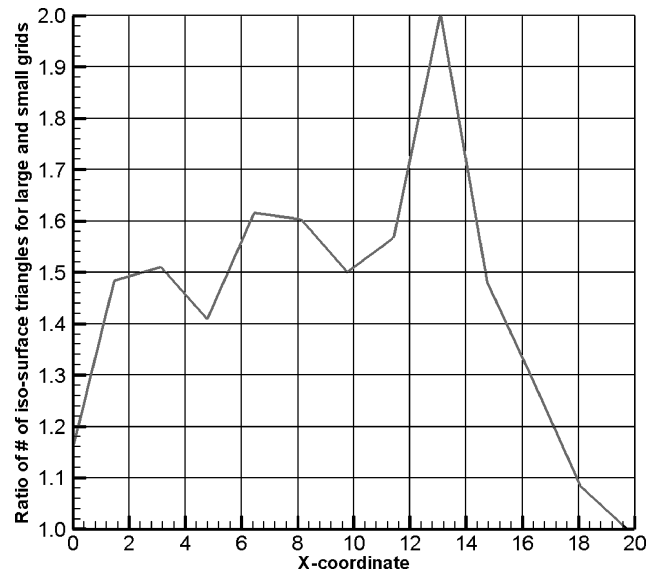


Fig. 10 Grid sensitivity analysis for isosurfaces.

GUI. This is also scalable because 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 postprocess the entire solution on a workstation (which is often not feasible).

Figure 11 shows such quantitative comparisons of experimental and computational results for a rigid rotor. The inviscid Euler equations have been solved in hover conditions⁵ using the flow solver PUMA2 with moving grids to simulate an experiment conducted by Caradonna and Tung.⁵⁴ Chen and McCroskey,⁵⁵ Srinivasan and Baeder,⁵⁶ Berkman et al.,⁵⁷ and Boelens et al.⁵⁸ have performed computational simulations for this rotor and had good comparisons with the experimental data that were used extensively for the validation and development of rotor aerodynamics and performance codes. Two unstructured grids were created for the two-bladed rotor with a NACA 0012 airfoil section. The first grid (grid 1) had 1.3 million tetrahedral cells with nearly homogenous mesh distribution on the surface of rotor blades (Fig. 12a). The second grid (grid 2) had 0.9 million cells, clustered in the leading-edge region (Fig. 12b) and in the region of tip vortices with increasing cell size toward the outer boundaries. The rotor simulation was run on 16 processors on the parallel personal computer cluster²⁸ COCOA-2, consisting of 40 800-MHz Pentium III processors and 20-GB RAM. The average memory consumed per processor was 110 MB for grid 1, and 83 MB for grid 2. The time-accurate computations for one revolution took nearly 15.5 days using an explicit two-stage Runge–Kutta method for grid 1. The thrust coefficient for grid 1 for a two-bladed rotor in hover has converged to the experimental value after nearly 3.4 revolutions, as shown in Fig. 13. The spanwise pressure coefficient distribution at the quarter-chord location and the chordwise pressure coefficient distribution at 80% spanwise station for both grids with the comparison to the experimental data are shown in Figs. 11a and 11b, respectively. Figures 11a and 11b are obtained using the Tecplot integration feature of the POSSE GUI as the simulation was being performed on COCOA-2 in parallel. The pressure distributions show good agreement with the experimental values. The calculated pressure deficiencies at the leading edge are due to insufficient grid resolution.

Visualization using Virtual Reality Systems

Visualizations with POSSE can be displayed using VR facilities (such as CAVes and RAVes) to better understand the three-dimensional nature of the flowfields.⁶ Complex fluid problems are extremely difficult to understand using conventional

two-dimensional graphics. Isosurfaces in three-dimensional space can be used to highlight some flow features, but these really need to be viewed using stereographics. Example applications of time-dependent and three-dimensional flow simulations performed by our research group are discussed hereafter to show the usefulness of POSSE and VR systems.

One of the complex flow simulations performed using the CFD solver PUMA2 is the unsteady airwake simulations over an LHA-

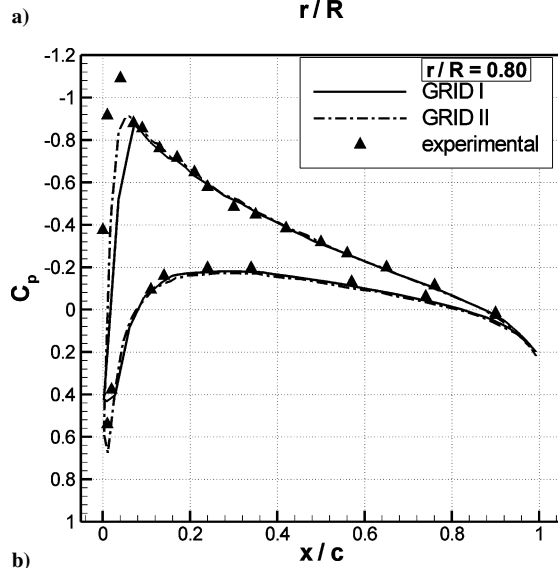
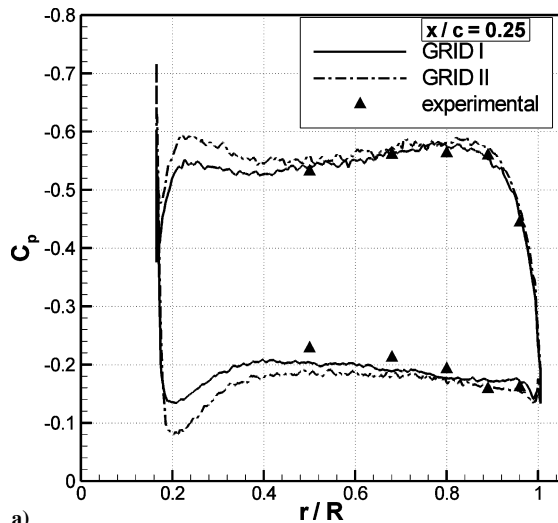
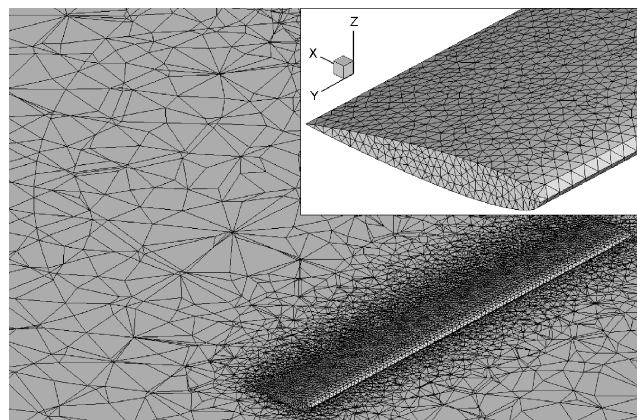


Fig. 11 Pressure coefficient distribution: a) spanwise at quarter-chord location and b) chordwise at 80% of radius.



a) Grid 1

class ship. The results of the steady-state and time-accurate simulations of the ship airwake, which a helicopter flight dynamics simulation model was interfaced with, have been discussed in detail by Lee et al.^{49,50} In this simulation, the time-accurate computations are started from the pseudo-steady-state solution, and the simulation time step ($480 \mu s$) is determined by the smallest cell size in the volume grid. It takes nearly 2080 iterations to compute 1 s of real flow (~ 1.8 h on 12 processors on the personal computer cluster LION-XL⁵⁹), with the current grid having 850,000 cells, and by the use of an explicit Runge–Kutta time integration scheme. A total of 50 s of time-accurate flow solution is obtained. Only the last 40 s of the time history data are stored for every 0.1 s to be used for the dynamic interface simulations for a UH-60A operating from the LHA. Each CFD flow solution file is 41 MB in size, whereas the velocity data at an instant required in dynamic interface simulation are only 5.2 MB. The extraction of the velocity field (for a smaller dynamic interface domain over selected landing spots) from the CFD data was performed by postprocessing the stored flow data. An instantaneous isosurface of vorticity magnitude over the LHA ship for both 0- and 30-deg yaw cases with 30 kn of relative wind speed are shown in Fig. 14. Complex flow features such as deck-edge vortices, separated vortical regions on the deck, and a complex island wake can be seen in Fig. 14. However, these unsteady flow features can be better observed by creating an animation of isosurfaces in time. Furthermore, better observations can be made if one can move around and inside the isosurfaces freely such as with an

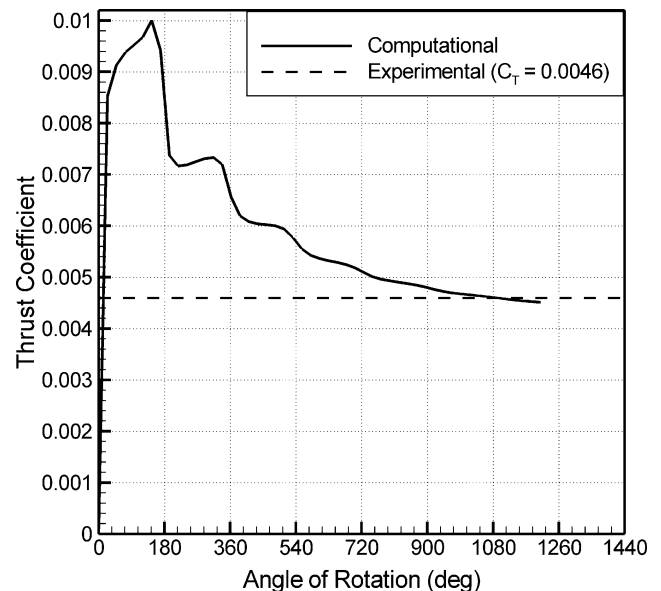
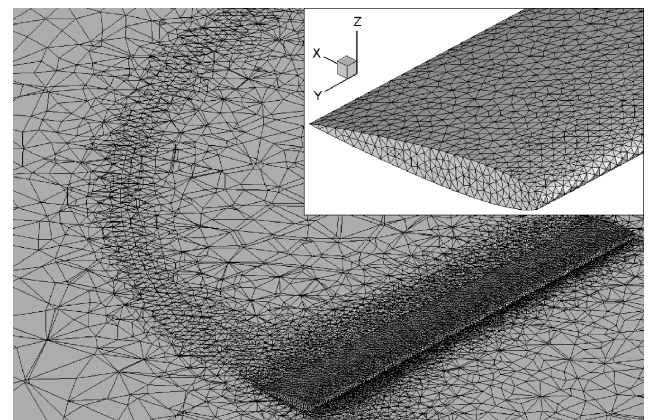


Fig. 13 Time history of the rotor thrust coefficient (grid 1).



b) Grid 2

Fig. 12 Unstructured mesh on the rectangular rotor blade and on the XY plane (at $z = 0$) for two grids.

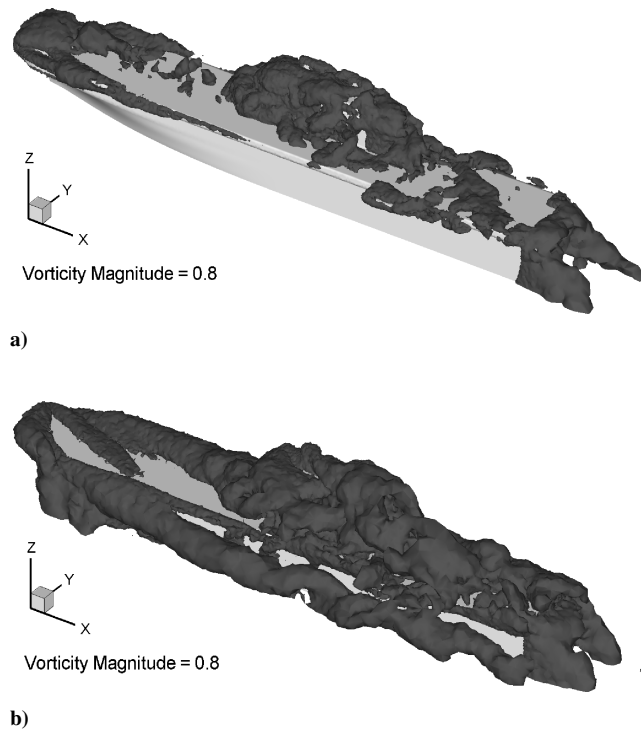


Fig. 14 Vorticity magnitude isosurface at $t = 40$ s for a) 0-deg yaw and b) 30-deg yaw cases over the LHA ship.⁴⁹

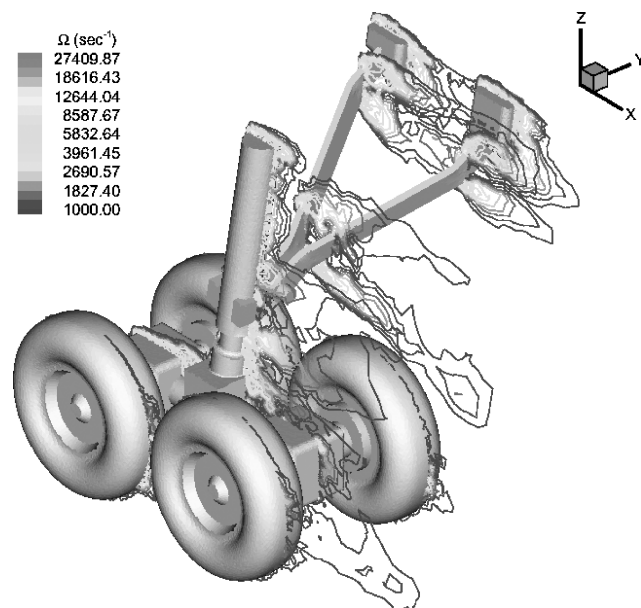


Fig. 15 Instantaneous isovorticity lines for landing gear.⁴⁴

interactive visualization in a VR system. POSSE was used in this simulation to observe the flow features at any intermediate iteration and also over different landing spots using the VR system as the simulation is running on COCOA-2. Stereographics displays are useful to visualize and better understand the details of such complex flows. Because the CFD computations take so much wall clock time, the real-time computations of the ship airwake and interface of the airwake with a flight dynamics model in real time, for example, in a helicopter simulator for pilot training purposes, is not yet possible.

Another example is the numerical simulation study of time-accurate flow and noise due to a detailed landing gear configuration using PUMA2. Souliez et al.⁴⁴ discussed the aerodynamic

and acoustic results of this unsteady simulation. Figure 15 shows a three-dimensional 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 three-dimensional flowfields change in time. In the future, simulations like these can be studied with stereographics combined with stereo sound.

Stereographics displays are invaluable for viewing and understanding the complex, three-dimensional nature of flows such as those in rotor wake, ship airwake, and flows around rotor fuselage and landing gear. 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. The use of POSSE in such wide range of large-scale and time-accurate aerospace simulations is important and necessary because it allows interactive, real-time computational steering and visualization in a scalable way.

Conclusions

POSSE, a general-purpose computational steering library, is a software system with a simple user interface. Scientists and engineers often need an easy-to-use software system such as POSSE. The advanced object-oriented features of C++ have given POSSE an edge over existing steering libraries written in older languages. POSSE has been successfully coupled to a C++ based flow solver PUMA2. It is very important to have libraries such as POSSE to visualize large-scale parallel simulations. Large parallel aerospace simulations need scalable visualization solutions such as these. Benefits of scalability and dimensional reduction arising from this approach were imperative in the development of this computational steering system. POSSE has been successfully tested with several flow simulations. For example, the results of a rotor flowfield computation in hover are compared with experimental measurements using the Tecplot integration feature of the POSSE GUI. In addition, several complex flow problems have been discussed that are extremely difficult to understand using conventional two-dimensional graphics. Isosurfaces in three-dimensional space can be used to highlight some flow features, but these really need to be viewed using stereographics. For fourth-dimensional problems that change in both space and time, it will be essential to use stereographics. At a more basic level, the ability to interact and visualize a complex solution as it unfolds, and the real-time nature of the computational steering system, opens a whole new dimension to the scientist and engineer for interacting with their simulations. POSSE is the best interactive computational steering and monitoring approach, and people who are performing such large-scale parallel aerospace simulations could use it or a similar approach.

Acknowledgments

This work was supported by National Science Foundation Grants DGE-9987589, EIA 99-77526, and ACI-9908057, U.S. Department of Energy Grant DG-FG02-99ER25373, the Alfred P. Sloan Foundation, and National Rotorcraft Technology Center: Rotorcraft Center of Excellence NGT 2-52275.

References

- ¹Pope, S. B., *Turbulent Flows*, Cambridge Univ. Press, Cambridge, England, U.K., 2001, Chap. 9.
- ²Kaneda, Y., Ishihara, T., Yokokawa, M., Itakura, K., and Uno, A., "Energy Dissipation Rate and Energy Spectrum in High Resolution Direct Numerical Simulations of Turbulence in a Periodic Box," *Physics of Fluids*, Vol. 15, No. 2, Feb. 2003, pp. L21-L24.
- ³Spalart, P. R., "Strategies for Turbulence Modelling and Simulations," *International Journal of Heat and Fluid Flow*, Vol. 21, No. 3, June 2000, pp. 252-263.

- ⁴Modi, A., "POSSE: Portable Object-oriented Scientific Steering Environment," URL: <http://posse.sourceforge.net>, 2001.
- ⁵Modi, A., Sezer-Uzol, N., Long, L. N., and Plassmann, P. E., "Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations," AIAA Paper 2002-2750, June 2002.
- ⁶Sezer-Uzol, N., Modi, A., Long, L. N., and Plassmann, P. E., "Visualizing Computational Simulation Results Using Virtual Reality Technology," *Proceedings of FEDSM'03, 4th ASME/JSME Joint Fluids Engineering Conference*, ASME, New York, July 2003.
- ⁷Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann Pub., San Francisco, 1997.
- ⁸Long, L. N., and Modi, A., "Turbulent Flow and Aeroacoustic Simulations Using a Cluster of Workstations," *NCSA Linux Revolution Conference*, June 2001.
- ⁹Sterling, T., Salmon, J., Becker, D. J., and Savarese, D. F., "How to Build a Beowulf," The MIT Press, Cambridge, MA, 1999.
- ¹⁰Modi, A., and Long, L. N., "Unsteady Separated Flow Simulations Using a Cluster of Workstations," AIAA Paper 2000-0272, Jan. 2000.
- ¹¹Fakespace Systems, RAVE, Fakespace Systems Inc., Marshalltown, Iowa, URL: <http://www.fakespace.com/>.
- ¹²Gu, 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 Computation*, IEEE Publications, Los Alamitos, CA, Jan. 1995, pp. 422-429.
- ¹³Parker, S., Miller, M., Hansen, C., and Johnson, C. R., "An Integrated Problem Solving Environment: The SCIRun Computational Steering System," *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS31)*, IEEE Computer Society, IEEE Publications, Los Alamitos, CA, Vol. 7, Jan. 1998, pp. 147-156.
- ¹⁴Ba, I., Malon, C., and Smith, B., "Design of the ALICE Memory Snooper," Argonne, IL, URL: <http://www.mcs.anl.gov/ams>, 1999.
- ¹⁵Jablonski, D., Bruner, J., Bliss, B., and Haber, R., "VASE: The Visualization and Application Steering Environment," *Proceedings of Supercomputing '93*, IEEE, Computer Society Press, Los Alamitos, CA, 1993, pp. 560-569.
- ¹⁶Geist, I. G. A., Kohl, J. A., and Papadopoulos, P. M., "CUMULVS: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications," *International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, No. 3, 1997, pp. 224-235.
- ¹⁷van Liere, R., and van Wijk, J. J., "CSE: A Modular Architecture for Computational Steering," *Proceedings of the 7th Eurographics Workshop on Visualization in Scientific Computing*, Springer-Verlag, Wien, April 1996, pp. 257-266.
- ¹⁸Haimes, R., "pV3: A Distributed System for Large-Scale Unsteady CFD Visualization," AIAA Paper 94-0321, 1994.
- ¹⁹Shaffer, E., Reed, D., Whitmore, S., and Schaeffer, B., "Virtue: Performance Visualization of Parallel and Distributed Applications," *IEEE Computer Graphics*, Vol. 32, No. 12, 1999, pp. 44-51.
- ²⁰"COVISE: Collaborative Visualization and Simulation Environment," High Performance Computing Center, Visualization Dept., Univ. of Stuttgart, Stuttgart, Germany, June 2004, URL: <http://www.hlr.de/organization/vis/covise/>.
- ²¹Rantau, D., and Lang, U., "A Scalable Virtual Environment for Large Scale Scientific Data Analysis," *Future Generation Computer Systems*, Vol. 14, No. 3-4, 1998, pp. 215-222.
- ²²Burnett, M., Hossli, R., Pulliam, T., VanVoorst, B., and Yang, X., "Toward Visual Programming Languages for Steering Scientific Computations," *IEEE Computational Science and Engineering*, Vol. 1, No. 4, 1994, pp. 44-62.
- ²³Mulder, J., van Wijk, J., and van Liere, R., "A Survey of Computational Steering Environments," *Future Generation Computer Systems*, Vol. 15, No. 1, 1999, pp. 119-129.
- ²⁴Parker, S. G., "The SCIRun Problem Solving Environment and Computational Steering Software System," Ph.D. Dissertation, Dept. of Computer Science, Univ. of Utah, Salt Lake City, UT, Aug. 1999.
- ²⁵Reitinger, B., "On-Line Program and Data Visualization of Parallel Systems in a Monitoring and Steering Environment," M.S. Thesis, Dept. for Graphics and Parallel Processing, Johannes Kepler Univ., Linz, Austria, Jan. 2001.
- ²⁶Cruz-Neira, C., Leigh, J., Pakpa, M., Barnes, C., Cohen, S. M., Das, S., Engelmann, R., Hudson, R., Roy, T., Siegel, L., Vasilakis, C., DeFanti, T. A., and Sandin, D. J., "Scientists in Wonderland: A Report on Visualization Application in the CAVE Virtual Reality Environment," *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*, IEEE, Los Alamitos, CA, Oct. 1993, pp. 59-66.
- ²⁷Goldman, J., and Roy, T. M., "The Cosmic Worm," *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, 1994, pp. 12-14.
- ²⁸Modi, A., "Software System Development for Real-Time Simulations Coupled to Virtual Reality for Aerospace Applications," Ph.D. Dissertation, Dept. of Computer Science and Engineering, Pennsylvania State Univ., University Park, PA, Aug. 2002.
- ²⁹Modi, A., Long, L. N., and Plassmann, P. E., "Real-Time Visualization of Wake-Vortex Simulations Using Computational Steering and Beowulf Clusters, in High Performance Computing for Computational Science," *VECPAR 2002, Lecture Notes in Computer Science*, edited by J. Palma, J. Dongarra, V. Hernandez, and A. de Sousa, No. 2565, Springer-Verlag, Berlin, 2002, pp. 464-478.
- ³⁰Arsenault, L., Kelso, J., Kriz, R., and Das-Neves, F. D., "DIVERSE: A Software Toolkit to Integrate Distributed Simulations with Heterogeneous Virtual Environments," *Computer Science Technical Report TR-01-10*, Virginia Tech, VA, 2001.
- ³¹Cruz-Neira, C., "Applied Virtual Reality," *Course 14 Notes, Proceedings of SIGGRAPH '98*, ACM Press, New York, July 1988.
- ³²Ensight, Computational Engineering International (CEI), Apex, North Carolina, URL: <http://www.ceintl.com/>.
- ³³Bruner, C. W. S., "Parallelization of the Euler Equations on Unstructured Grids," Ph.D. Dissertation, Dept. of Aerospace Engineering, Virginia Polytechnic Inst. and State Univ., Blacksburg, VA, May 1996.
- ³⁴Thompson, P. A., *Compressible Fluid Dynamics*, McGraw-Hill, New York, 1971.
- ³⁵Souliez, F. J., "Parallel Methods for the Computation of Unsteady Separated Flows Around Complex Geometries," Ph.D. Dissertation, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, Aug. 2002.
- ³⁶Souliez, F. J., Long, L. N., Morris, P. J., and Sharma, A., "Landing Gear Aerodynamic Noise Prediction Using Unstructured Grids," *International Journal of Aeroacoustics*, Vol. 1, No. 2, 2002, pp. 115-135.
- ³⁷Modi, A., "Unsteady Separated Flow Simulations Using a Cluster of Workstations," M.S. Thesis, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, May 1999.
- ³⁸Savary, N., "A Higher-Order Accurate Finite Volume Method Using Unstructured Grids for Unsteady Fluid Dynamics," M.S. Thesis, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, Dec. 1999.
- ³⁹Schweitzer, S., "Computational Simulation of Flow Around Helicopter Fuselages," M.S. Thesis, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, May 1999.
- ⁴⁰Sezer-Uzol, N., "High Accuracy Wake and Vortex Simulations Using a Hybrid Euler/Discrete Vortex Method," M.S. Thesis, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, May 2001.
- ⁴¹Sezer-Uzol, N., and Long, L. N., "High-Accuracy Wake and Vortex Simulations Using a Hybrid Euler/Discrete Vortex Method," AIAA Paper 2000-2029, June 2000.
- ⁴²Sharma, A., "Parallel Methods for Unsteady, Separated Flows and Aerodynamic Noise Prediction," M.S. Thesis, Dept. of Aerospace Engineering, Pennsylvania State Univ., University Park, PA, Aug. 2001.
- ⁴³Sharma, A., and Long, L. N., "Airwake Simulations on an LPD 17 Ship," AIAA Paper 2001-2589, June 2001.
- ⁴⁴Souliez, F. J., Long, L. N., Morris, P. J., and Sharma, A., "Landing Gear Aerodynamics Noise Prediction Using Unstructured Grids," *International Journal of Aeroacoustics*, Vol. 1, No. 2, 2002, pp. 115-135.
- ⁴⁵Lorensen, W. E., and Cline, H. E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *SIGGRAPH '87 Proceedings*, ACM Press, New York, Vol. 21, No. 4, 1987, pp. 163-169.
- ⁴⁶Guezec, A., and Hummel, R., "Exploiting Triangulated Surface Extraction Using Tetrahedral Decomposition," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 4, Dec. 1995, pp. 328-342.
- ⁴⁷"Fast Light ToolKit (FLTK)," URL: <http://www.fltk.org>, 4 June 2004.
- ⁴⁸"Visualization ToolKit (VTK)," URL: <http://www.kitware.com/vtk.html>, 4 June 2004.
- ⁴⁹Lee, D., Horn, J. F., Sezer-Uzol, N., and Long, L. N., "Simulation of Pilot Control Activity During Helicopter Shipboard Operation," AIAA Paper 2003-5306, Aug. 2003.
- ⁵⁰Lee, D., Sezer-Uzol, N., Horn, J. F., and Long, L. N., "Simulation of Helicopter Shipboard Launch and Recovery with Time-Accurate Airwaves," *Proceedings of the 59th American Helicopter Society Annual Forum*, American Helicopter Society, Alexandria, VA, May 2003.
- ⁵¹Tecplot, Tecplot Inc., Bellevue, WA, URL: <http://www.amtec.com/>.
- ⁵²Turnage, A., "Reducing Aircraft Noise with Computer Graphics," *IEEE Computer Graphics and Applications*, Vol. 22, No. 3, May 2002, pp. 16-21.

⁵³Fieldview, Intelligent Light, Rutherford, NJ, URL: <http://www.ilight.com/>, 4 June 2004.

⁵⁴Caradonna, F. X., and Tung, C., "Experimental and Analytical Studies of a Model Helicopter Rotor in Hover," *Vertica*, Vol. 5, No. 2, 1981, pp. 149–161.

⁵⁵Chen, C. L., and McCroskey, W. J., "Numerical Simulation of Helicopter Multi-Bladed Rotor Flow," AIAA Paper 88-0046, Jan. 1988.

⁵⁶Srinivasan, G. R., and Baeder, J. D., "TURNS: A Free-Wake Euler/Navier–Stokes Numerical Method for Helicopter Rotors," *AIAA Jour-*

nal, Vol. 31, No. 5, 1993, pp. 959–962.

⁵⁷Berkman, M. E., Sankar, L. N., Berezin, C. R., and Torok, M. S., "Navier–Stokes/Full-Potential/Free-Wake Method for Rotor Flows," *Journal of Aircraft*, Vol. 34, No. 5, 1997, pp. 635–640.

⁵⁸Boelens, O. J., van der Ven, H., Oskam, B., and Hassan, A., "Accurate and Efficient Vortex-Capturing for Rotor Blade–Vortex Interaction," AIAA Paper 2000-0112, Jan. 2000.

⁵⁹LION-XL PC Cluster, High Performance Computing Group, Pennsylvania State University, University Park, PA, URL: <http://gears.aset.psu.edu/hpc/systems/lionxl/>, 4 June 2004.