AIAA 2002-2750

Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations

Anirudh Modi, Nilay Sezer-Uzol, Lyle N. Long, and Paul E. Plassmann
The Pennsylvania State University
University Park, PA

32nd AIAA Fluid Dynamics Conference and Exhibit
24 – 27 June 2002 / St. Louis, Missouri

AIAA-2002-2750

Scalable Computational Steering System for Visualization of Large-Scale CFD Simulations

Anirudh Modi*, Nilay Sezer-Uzol**, Lyle N. Long†, Paul E. Plassmann††
The Pennsylvania State University
University Park, PA 16802

ABSTRACT

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 Navier-Stokes flow solver (PUMA2). This paper illustrates how to use "computational steering" with PUMA2 to visualize CFD solutions while they are being computed, and even change the input data while it is running. 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 or Beowulf clusters, while the visualization is performed on other computers, through a client-server approach. A key advantage of our system is its scalability. The visualization is performed using a parallel approach. 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 solutions from this solver are presented to show the usefulness of POSSE. The examples include unsteady ship airwake simulations, unsteady flow over a helicopter fuselage, and unsteady simulations of a helicopter rotor. The results of the rotor simulations in hover are compared with the experimental measurement and discussed in some detail. The advantages of using object-oriented programming are also discussed.

* Ph.D. Candidate, Department of Computer Science and Engineering
** Ph.D. Candidate, Department of Aerospace Engineering
† Professor, Department of Aerospace Engineering, Associate Fellow AIAA
†† Associate Professor, Department of Computer Science and Engineering

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 integrated with such CFD software. At Penn State, an easy to use, general-purpose computational steering library: *Portable Object-oriented Scientific Steering Environment (POSSE)*, has been developed which can be easily coupled to any existing C/C++ simulation code. C++ has several advantages over FORTRAN making the use of the latter difficult and unnecessary in today's environment.

COMPUTATIONAL MONITORING AND STEERING LIBRARY

The computational monitoring and steering library, POSSE[1], 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 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 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),[2,3,4] has been coupled to our Virtual Reality (VR) system, a Fakespace Reconfigurable Automatic Virtual Environment (RAVE),[5] 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 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.

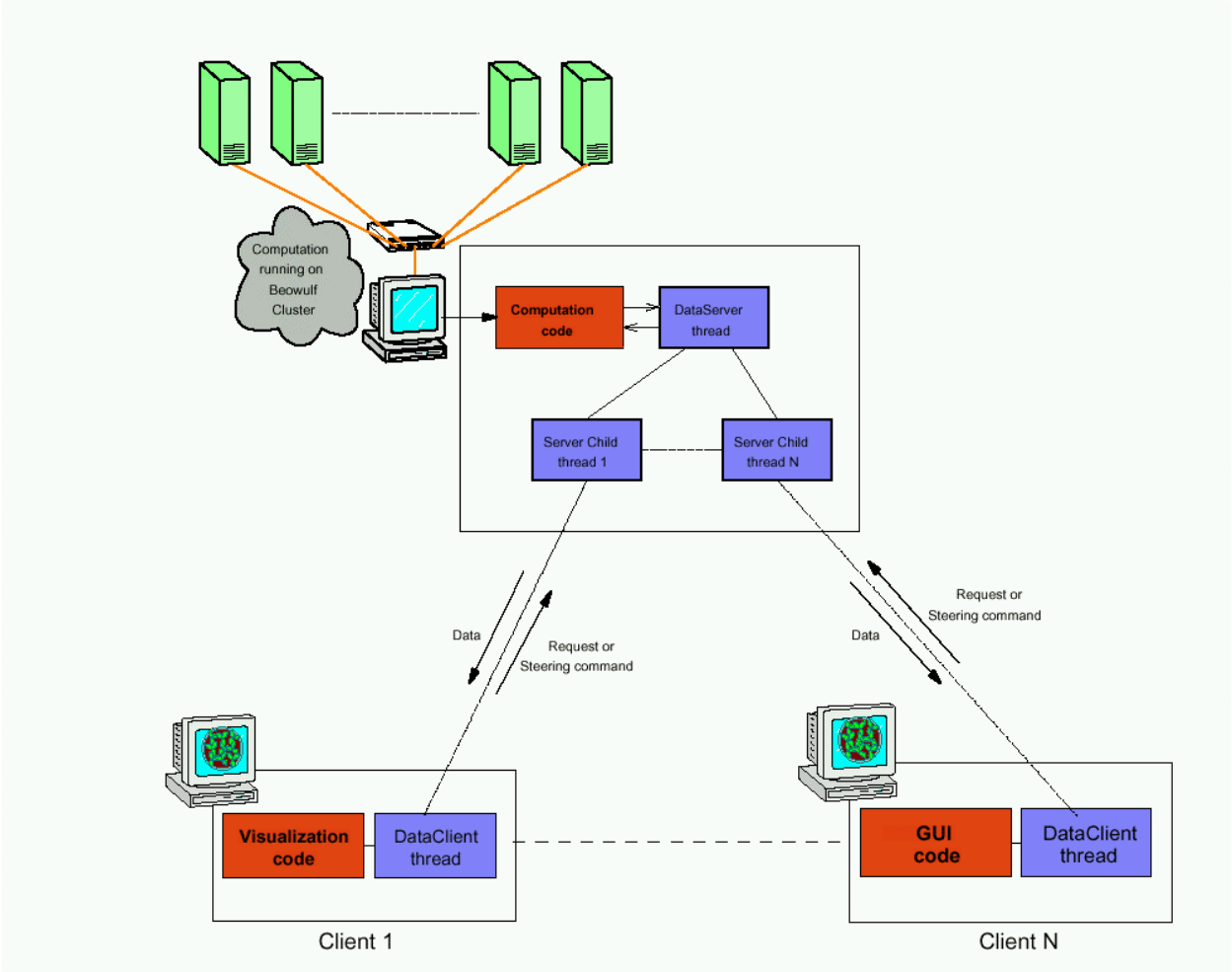


Figure 1. A Schematic view of POSSE

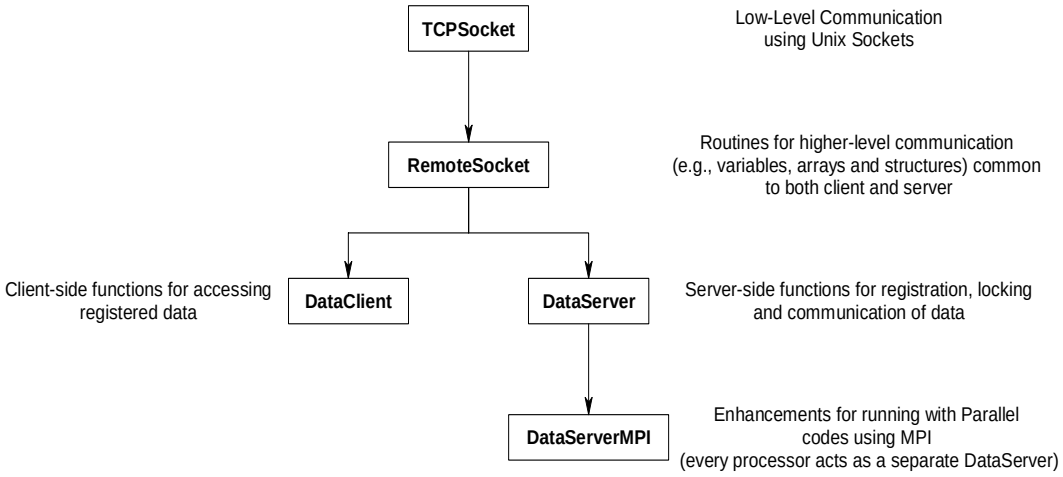


Figure 2. Inheritance diagram for POSSE

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[6] (an on-line monitoring and steering toolkit developed at Georgia Tech) and ALICE Memory Snooper[7] (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. 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 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. Figure 1 shows a schematic view of how POSSE can be used, and Figure 2 shows the inheritance diagram of the POSSE classes. As seen in Figure 1, 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 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.

POSSE is designed to be extremely lightweight, portable and efficient. It 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 multi-threaded, supporting several clients simultaneously. It can also be easily incorporated into parallel simulations based on the Message Passing Interface (MPI)[8] 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 3 and Figure 4 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.[9,10]

```
//---------------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 3. 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 4. A simple, complete POSSE client application written in C++

3-D FINITE VOLUME CFD SOLVER

PUMA2 is the modified version of the flow solver PUMA (Parallel Unstructured Maritime Aerodynamics CFD solver)[11] which uses the finite volume formulation of the Navier Stokes equations for 3-D, internal and external, non-reacting, compressible, unsteady or steady state solutions of problems for complex geometries. Penn State has been refining and developing this code since 1997.[10,12-23] The integral form of the Navier-Stokes equations are

$$\frac{\partial}{\partial t}\int_V \mathbf{Q}dV + \oint_S (\mathrm{F}\cdot\mathbf{n})dS - \oint_S (\mathrm{F}_v\cdot\mathbf{n})dS = 0 \quad (2)$$

$$\mathbf{F} = \mathbf{F}_x + \mathbf{F}_y + \mathbf{F}_z$$

$$\mathbf{F}_v = \mathbf{F}_{vx} + \mathbf{F}_{vy} + \mathbf{F}_{vz}$$

$$\mathbf{Q} = \begin{bmatrix}\rho & \rho u & \rho v & \rho w & \rho e_0\end{bmatrix}^T$$

$$\mathbf{F}_x = \begin{Bmatrix} \rho(u - b_x) \\ \rho u(u - b_x) + p \\ \rho v(u - b_x) \\ \rho w(u - b_x) \\ \rho h_0(u - b_x) + p b_x \end{Bmatrix}$$

$$\mathbf{F}_y = \begin{Bmatrix} \rho(v - b_y) \\ \rho u(v - b_y) \\ \rho v(v - b_y) + p \\ \rho w(v - b_y) \\ \rho h_0(v - b_y) + p b_y \end{Bmatrix}$$

$$\mathbf{F}_z = \begin{Bmatrix} \rho(w - b_z) \\ \rho u(w - b_z) \\ \rho v(w - b_z) \\ \rho w(w - b_z) + p \\ \rho h_0(w - b_z) + p b_z \end{Bmatrix}$$

$$\mathbf{F}_{vx} = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ (u\tau_{xx} + v\tau_{xy} + w\tau_{xz}) - q_x \end{Bmatrix}$$

$$\mathbf{F}_{vy} = \begin{Bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ (u\tau_{yx} + v\tau_{yy} + w\tau_{yz}) - q_y \end{Bmatrix}$$

$$\mathbf{F}_{vz} = \begin{Bmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ (u\tau_{xz} + v\tau_{yz} + w\tau_{zz}) - q_z \end{Bmatrix}$$

where **F** is the inviscid fluxes with the rotational terms[24] and $\mathbf{F}_v$ is the viscous flux terms. **U**(u,v,w) is the absolute flow velocity and **b**(b_x,b_y,b_z) is the grid velocity. Pressure, total energy and total enthalpy are given by

$$p = (\gamma - 1)\rho e$$

$$e_0 = e + \frac{1}{2}\mathbf{U} \cdot \mathbf{U}$$

$$h_0 = e_0 + \frac{p}{\rho}$$

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. PUMA2 can be run so as to preserve time accuracy or as a pseudo-unsteady 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 has been developed to solve 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 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.

Unsteady Rotor Flow simulations

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

Inviscid Euler computations are performed in hover conditions with the flow solver PUMA2 to simulate an

experiment conducted by Caradonna & Tung[25] who have performed a benchmark test specifically to aid in the development of rotor performance codes. A rectangular, untwisted and untapered two bladed rotor model with NACA 0012 cross section and an aspect ratio of 6 was used in the experiments. Blade pressure measurements and tip vortex surveys (vortex strength and geometry) were obtained for a wide range of tip Mach numbers. The computational rotor geometry has two blades with 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.

The 3-D unstructured grid for the two bladed rotor geometry is generated using the GRIDGEN software system of Pointwise Inc.[26] for the generation of (3-D) grids and meshes. The cylindrical computational grid 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. Figures 5 and 6 show the computational domain and the rotor blade surface mesh.

The rotational speed is chosen as 25 rad/sec to simulate an experimental hover case of 8 degrees pitch with 1250 RPM and tip Mach number of 0.439. The 2-stage Runge-Kutta time integration method with a CFL of 0.9 and with Roe's numerical flux scheme is used in the computations which are performed on the 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 time history of the rotor thrust coefficient, $C_T = T/[\rho A(\Omega R)^2]$, for two bladed rotor in hover is shown in Figure 7.

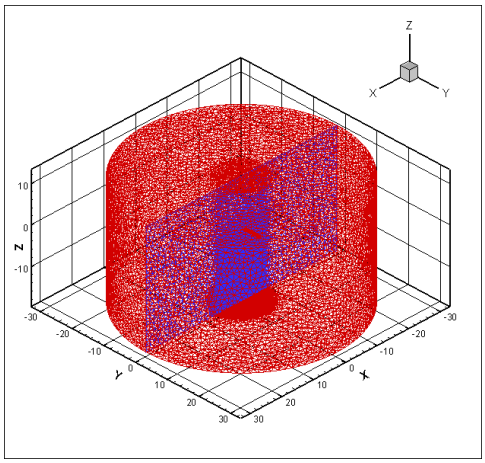


Figure 5. Unstructured grid domain used in computations

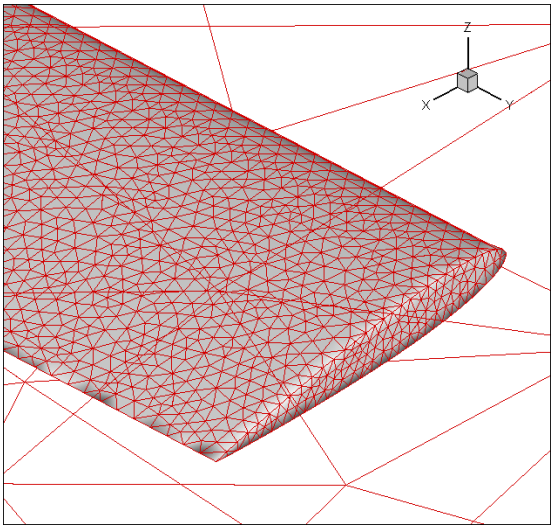


Figure 6. Unstructured rotor blade surface mesh

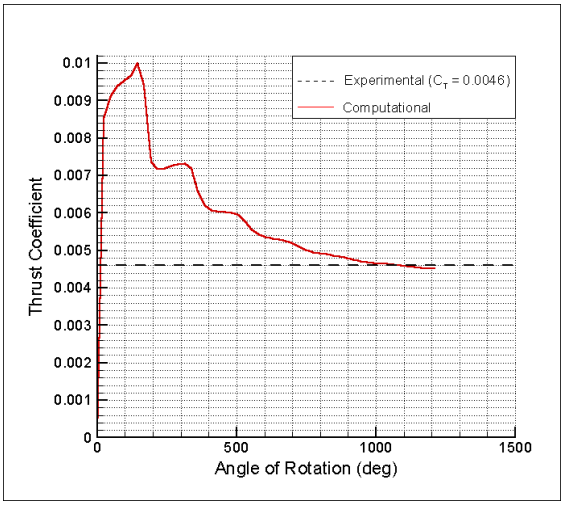


Figure 7. Time history of the rotor thrust coefficient

MODIFICATIONS TO PUMA2

To achieve the interactivity, 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 *[ρ , u, v, w, p]* and various important flow parameters in the code. Several new global variables were added to receive iso-surface requests and store resulting iso-surfaces. An iso-surface extraction routine also had to be added to PUMA2. Due to the use of unstructured mesh data consisting of tetrahedra, a variation of the classic "marching cubes" algorithm[27] is used for iso-surface extraction. The implementation is closely related to the marching cube algorithm except that the fundamental sampling structure here is a tetrahedron[28] instead of a cube. Since this implementation expects the flow data at the nodes

of every tetrahedron, a subroutine to interpolate the flow data from cell-centers to the nodes also had to be added to PUMA2.

Iso-surface Routines

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 literally billions of grid points. 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.

To get the iso-surfaces these steps are performed:

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

This is scalable to very large grids and thousands of processors. 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.

GRAPHICAL USER INTERFACE

A graphical user interface (GUI) client is used to connect to the computational steering server. Figure 8 show a screenshot from the POSSE GUI client application showing PUMA2 registered data. PUMA2 specific client is used to extract and visualize iso-surfaces. The client is written in C++ with the cross-platform FLTK[29] API for the GUI and the Visualization ToolKit (VTK)[30] 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 9 shows a screenshot of the VTK output on a separate window.

A drop-down menu is provided to choose the flow variable for which iso-surfaces are requested. After the numerical value for the iso-surface has been selected, a request is sent to the flow solver which responds by extracting the iso-surface for the given flow parameters on each of the processors; then collecting it on the master processor and sending the final iso-surface to the client. There are two modes provided for querying iso-surfaces. In the default mode, the user queries for iso-surfaces while the flow-solver is computing the solution for the next iteration. This mode can be slow if the user wants to query several iso-surfaces one after another, as 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 its CPU cycles to answering the client iso-surface requests without any lag. There is also an option "Get Grid", 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. Figures 9, 10 and 11 show iso-surfaces for a helicopter fuselage flow[21], a helicopter rotor flow[16,23] and a ship airwake flow[23], respectively.

POSSE Client Graphical User Interface

Server: cocoa.ihpca.psu.edu Port: 4444 Disconnect! Refresh Interval (sec): 5 About

M 0.18 Get It! Query Mode: ON Get Grid

Layout file: contour.lay Contour Surface Grid

Keyword	Description	Type -> Size	# of Elements	Value
MPI_nprocs	Variable	int -> 4	1	12
MPI_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
inpfile	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

Figure 8. A POSSE GUI to connect to the flow solver

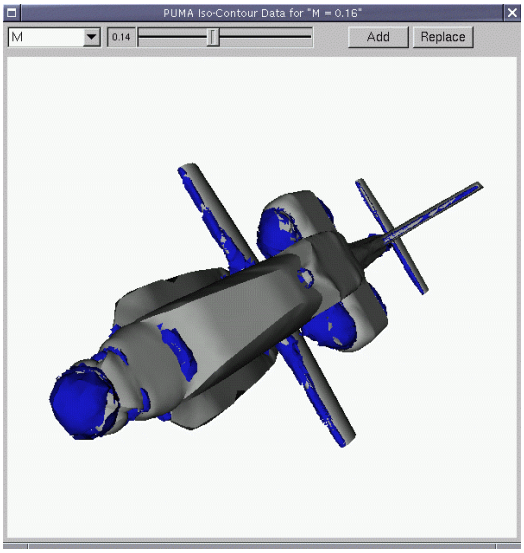


Figure 9. The POSSE Client application depicting iso-surfaces for a flow solution over the Apache helicopter

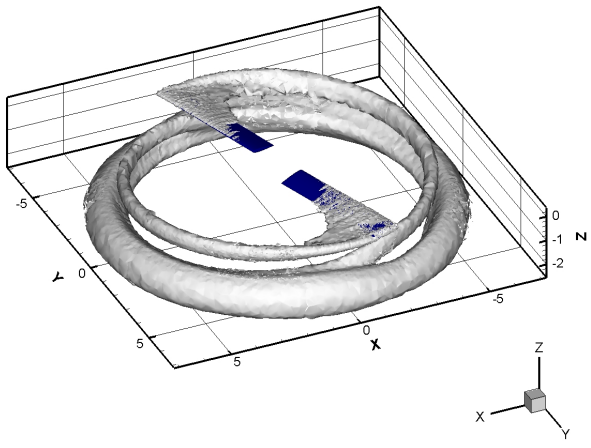


Figure 10. Entropy iso-sufaces for a flow solution over the rotor blades

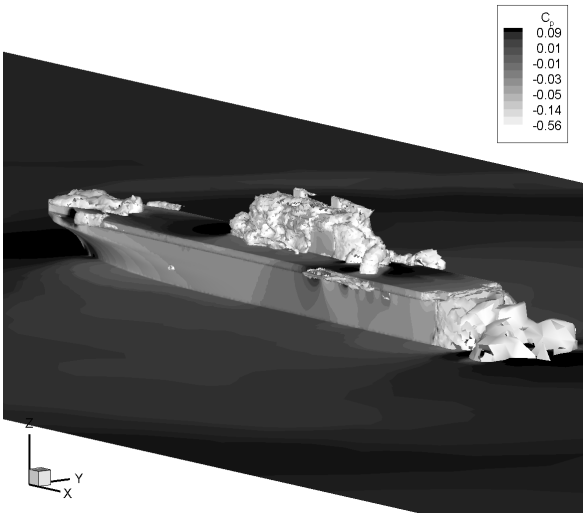


Figure 11. Entropy iso-surfaces and C_p surface contours for a flow solution over the LHA ship geometry

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 points on each processor of a parallel computation is *N/P* where *N* is the total number of grid faces and *P* is the total number of processors. Here 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. Thus the computational time for extraction of an iso-surface is *O(N/P)* as compared to the sequential algorithm which takes *O(N)* for the same procedure. 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). The total number of grid faces is $O(n^3)$ where *n* is the average number of grid faces in each direction. On the other hand, the number of triangles in an iso-surface obtained from this grid are only $O(n^2)$, which is an order of magnitude less data. Scalability and dimensional reduction combine to give us an expected *O(n/P)* data coming from each processor during the parallel computation of an iso-surface.

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. Figures 12, 13 and 14 show the relative size of the iso-surfaces 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. It can be clearly seen that the average number of triangles in an iso-surface is less than 1% of the total number of grid faces for this case. Figure 15 depicts another case where two grids with vastly varying sizes are used for extraction of iso-surfaces with varying values of X-coordinate. Here, although the larger grid (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 iso-surfaces 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 obtained above.

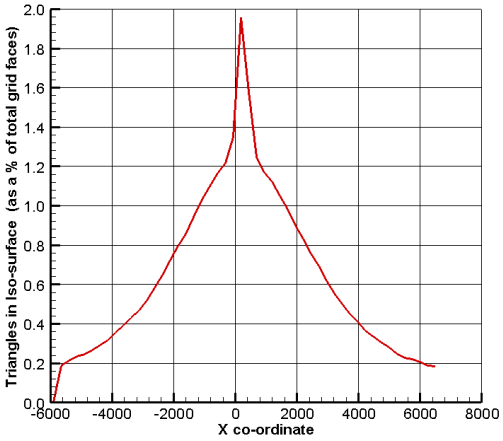


Figure 12. Percentage of X-coordinate iso-surface triangles for the Apache case

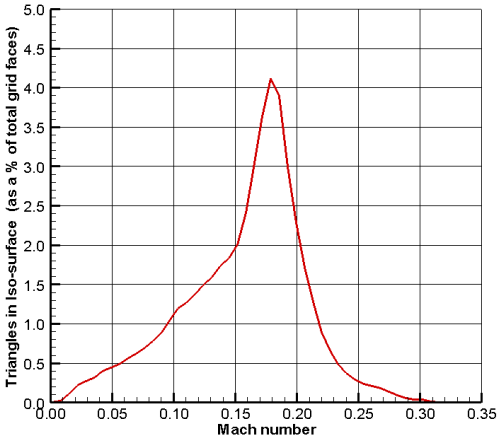


Figure 13. Percentage of Mach iso-surface triangles for the Apache case

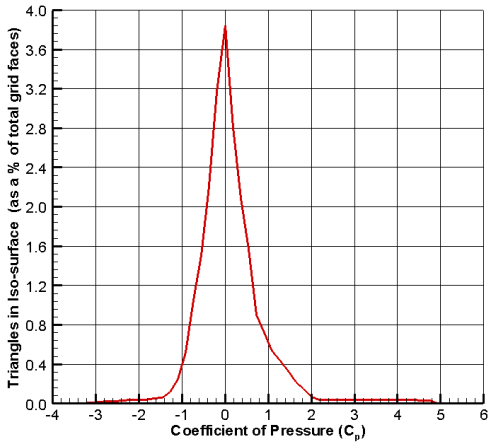


Figure 14. Percentage of C_p iso-surface triangles for the Apache case

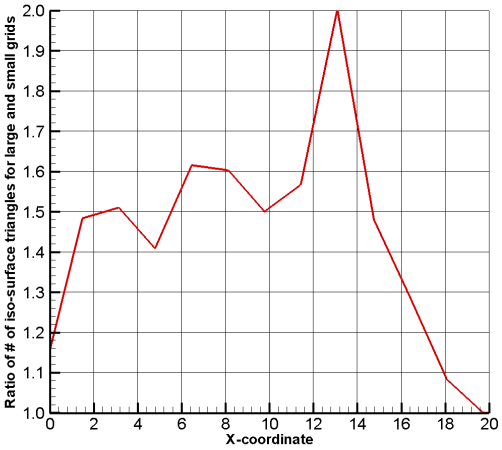


Figure 15. Grid sensitivity analysis for iso-surfaces

COUPLING OF THE SYSTEM TO OFF-THE-SHELF VISUALIZATION SOFTWARE

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[31], Ensight[32,33], or Fieldview[34]. 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 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).

The computational spanwise pressure coefficient distribution at the quarter chord location, the chordwise pressure coefficient distributions at several spanwise stations and the comparison with the experiments are shown in Figures 16 and 17. Although more computations need to be done for total convergence in hover case, the pressure distributions showed good agreement with the experimental values. The calculated pressure deficiencies at the leading edge are due to insufficient grid resolution.

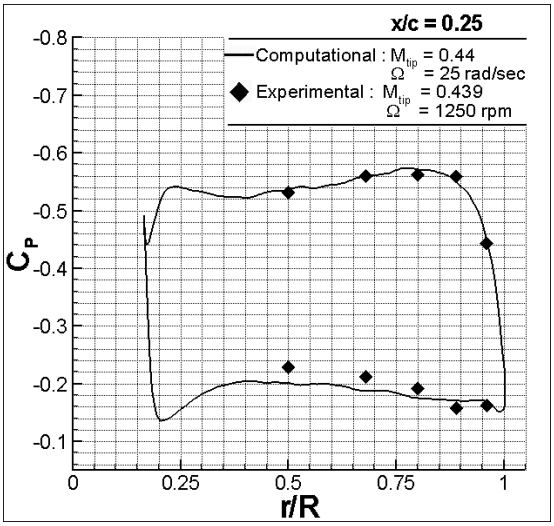
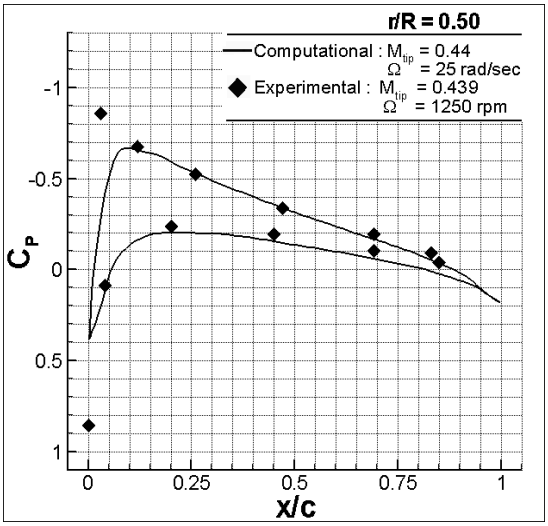
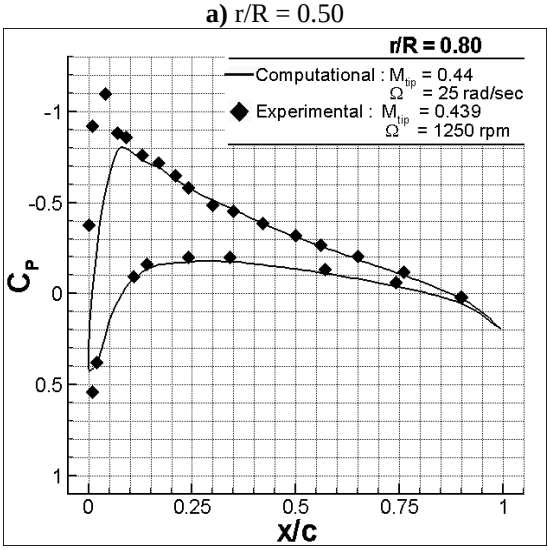


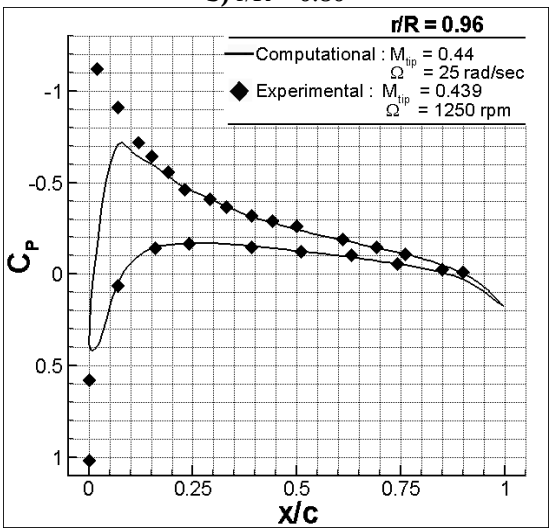
Figure 16. Spanwise pressure coefficient distribution at quarter chord location



a) r/R = 0.50



b) r/R = 0.80



c) r/R = 0.96

Figure 17. Chordwise pressure coefficient distributions at several spanwise stations

CONCLUSION

POSSE, a general-purpose computational steering system, is a powerful software system with a simple and elegant interface. Scientists and engineers 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 systems written in older languages. POSSE has been successfully coupled to a C-based flow solver PUMA2. Benefits of scalability and dimensional reduction arising from this approach were imperative in the development of this system. It 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. 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.

ACKNOWLEDGMENTS

This work was supported by NSF grants DGE-9987589, EIA 99-77526 and ACI-9908057, DOE grant DG-FG02-99ER25373, the Alfred P. Sloan Foundation, and RCOE NGT 2-52275.

REFERENCES

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

[2] 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.

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

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

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

[6] 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 Computation*, pp. 433–429, February 1995.

[7] Ba, I., Malon C., Smith, B., "Design of the ALICE Memory Snooper," http://www.mcs.anl.gov/ams, 1999.
[8] Peter S. Pacheco, "Parallel programming with MPI," Morgan Kaufmann Publishers, 1997
[9] Modi, A., Long, L.N., and Plassmann, P.E., "Real-Time Visualization of Vortex-Wake Simulations using Computational Steering and Beowulf Clusters," *VECPAR 200*2, June 2002, Lisbon, Portugal.
[10] 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
[11] Bruner, C.W.S., "Parallelization of the Euler Equations on Unstructured Grids," *PhD Dissertation, Virginia Polytechnic Institute and State University*, May 1996.
[12] Schweitzer, S., "Computational Simulation of Flow around Helicopter Fuselages," *M.S. Thesis, Dept. of Aerospace Engineering, The Pennsylvania State University*, May 1999.
[13] Modi, A., "Unsteady Separated Flow Simulations Using a Cluster of Workstations," *M.S. Thesis, Aerospace Engineering. The Pennsylvania State University*, May 1999.
[14] Savary, N., "A Higher-Order Accurate Finite Volume Method Using Unstructured Grids for Unsteady Fluid Dynamics," *M.S. Thesis, Dept. of Aerospace Engineering, The Pennsylvania State University*, December 1999.
[15] Sezer-Uzol, N. and Long, L. N., "High-Accuracy Wake and Vortex Simulations using a Hybrid CFD / DV Method," *AIAA Paper 2000-2029, 6th AIAA/CEAS Aeroacustics Conference*, June 12-14, 2000, Maui, Hawaii.
[16] 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.
[17] Sharma, A., and Long, L. N., "Airwake Simulations on an LPD 17 Ship," *AIAA Paper 2001-2589, Presented at 15th AIAA Computational Fluid Dynamics Conference*, June 11-14, 2001, Anaheim, California.
[18] Long, L.N., Souliez, F.J., Sharma, A., "Aerodynamic Noise Prediction Using Parallel Methods on Unstructured Grids," *AIAA Paper 2001-2196, 7th AIAA/CEAS Aeroacoustics Conference*, Maastricht, 2001, The Netherlands.
[19] Sharma, A., "Parallel Methods for Unsteady, Separated Flows and Aerodynamic Noise Prediction," *M.S. Thesis, Dept. of Aerospace Engineering, The Pennsylvania State University*, August 2001.
[20] 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)
[21] 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.
[22] Souliez, F.J. and Long, L. N., "Steady and Unsteady Flow Simulations Around a Bell 214ST Fuselage Using Unstructured Grids," *Submitted to the AHS Journal*.
[23] Sezer-Uzol, N., "Tiltrotor and Ship Airwake Flowfield Simulations," *PhD Thesis, Dept. of Aerospace Engineering, The Pennsylvania State University*, expected December 2003.
[24] Thompson, P.A., "Compressible Fluid Dynamics," McGraw-Hill, Inc., 1971.
[25] 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.
[26] Gridgen, http://pointwise.com
[27] Lorensen, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Agorithm," *SIGGRAPH '87 Proceedings*, Vol. 21, July 1987.
[28] Gueziec, A. and Hummel, R., "Exploiting Triangulated Surface Extraction using Tetrahedral Decomposition," *IEEE Transactions on Visualisation and Computer Graphics*, Vol. 1, No. 4, December 1995.
[29] The Fast Light ToolKit (FLTK), http://www.fltk.org, 1998.
[30] The Visualization ToolKit (VTK), http://www.kitware.com/vtk.html, 2000.
[31] Tecplot, http://www.amtec.com/
[32] Ensight, http://www.ceintl.com/
[33] Turnage, A., "Reducing Aircraft Noise with Computer Graphics," IEEE Computer Graphics, pp 16-21, May 2002.
[34] Fieldview, http://www.ilight.com/