Naval Surface Warfare Center
Carderock Division
West Bethesda, MD 20817-5700

NSWCCD-50-TR–2011/025 April 2011

Hydromechanics Department Report

Guide to NavyFOAM V1.0

by

Hua Shan, Keegan Delaney, Sung-Eun Kim, Bong Rhee, Joseph Gorski and Michael Ebert

NSWCCD-50-TR–2011/025 Guide to NavyFOAM V1.0

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
14-Mar-2011	Final	-

4. TITLE AND SUBTITLE

Guide to NavyFOAM V1.0

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hua Shan, Keegan Delaney, Sung-Eun Kim, Bong Rhee, Joseph Gorski and Michael Ebert

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
11-1-5705-411

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)

Naval Surface Warfare Center
Carderock Division
9500 MacArthur Boulevard
West Bethesda, MD 20817-5700

8. PERFORMING ORGANIZATION REPORT NUMBER

NSWCCD-50-TR-2011/025

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DoD HPC Modernization Program
Office
Attn: Dr. Douglass Post
10501 Furnace Road
Lorton, VA 22079

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release: Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes NavyFOAM V1.0, a computational fluid dynamics (CFD) capability based on Reynolds-averaged Navier-Stokes equations (RANSE) aimed at predicting turbulent single- and two-phase flows around ship hulls. The CFD capability employs a finite-volume discretization that allows use of arbitrary polyhedral elements. The free surface is captured using a volume-fraction method capable of accurately resolving sharp interfaces. NavyFOAM has been developed using an open-source CFD software tool-kit (OpenFOAM) that draws heavily upon object-oriented programming. The numerical methods and the physical models in the original version of OpenFOAM have been upgraded in an effort to improve accuracy and robustness of numerical solutions. The details of NavyFOAM V1.0 including the numerical methods and the physical models are described in this report. NavyFOAM V1.0 is demonstrated for a number of flows including: underwater bodies, turbulent free surface flows around the DTMB 5415 model and the KVLCC2 double-model. It is shown that the RANSE based approach can predict, with good accuracy, most of the salient features of the turbulent free-surface flows around the subject hulls including resistance, wave elevation, hull boundary layer and wake.

15. SUBJECT TERMS

Reynolds Averaged Navier-Stokes (RANS), computational fluid dynamics (CFD), NavyFOAM, OpenFOAM, Object Oriented Programming (OOP)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Hua Shan
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	142	19b. TELEPHONE NUMBER *(include area code)* 301-227-0573

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
14-Mar-2011	Final	-

4. TITLE AND SUBTITLE

Guide to NavyFOAM V1.0

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hua Shan, Keegan Delaney, Sung-Eun Kim, Bong Rhee, Joseph Gorski and Michael Ebert

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
11-1-5705-411

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AND ADDRESS(ES)

Naval Surface Warfare Center
Carderock Division
9500 MacArthur Boulevard
West Bethesda, MD 20817-5700

8. PERFORMING ORGANIZATION REPORT NUMBER

NSWCCD-50-TR-2011/025

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DoD HPC Modernization Program
Office
Attn: Dr. Douglass Post
10501 Furnace Road
Lorton, VA 22079

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release: Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes NavyFOAM V1.0, a computational fluid dynamics (CFD) capability based on Reynolds-averaged Navier-Stokes equations (RANSE) aimed at predicting turbulent single- and two-phase flows around ship hulls. The CFD capability employs a finite-volume discretization that allows use of arbitrary polyhedral elements. The free surface is captured using a volume-fraction method capable of accurately resolving sharp interfaces. NavyFOAM has been developed using an open-source CFD software tool-kit (OpenFOAM) that draws heavily upon object-oriented programming. The numerical methods and the physical models in the original version of OpenFOAM have been upgraded in an effort to improve accuracy and robustness of numerical solutions. The details of NavyFOAM V1.0 including the numerical methods and the physical models are described in this report. NavyFOAM V1.0 is demonstrated for a number of flows including: underwater bodies, turbulent free surface flows around the DTMB 5415 model and the KVLCC2 double-model. It is shown that the RANSE based approach can predict, with good accuracy, most of the salient features of the turbulent free-surface flows around the subject hulls including resistance, wave elevation, hull boundary layer and wake.

15. SUBJECT TERMS

Reynolds Averaged Navier-Stokes (RANS), computational fluid dynamics (CFD), NavyFOAM, OpenFOAM, Object Oriented Programming (OOP)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Hua Shan
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	142	19b. TELEPHONE NUMBER *(include area code)* 301-227-0573

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Contents

Figures

Tables

Page

THIS PAGE INTENTIONALLY LEFT BLANK

Abstract

This report describes NavyFOAM V1.0, a computational fluid dynamics (CFD) capability based on Reynolds-averaged Navier-Stokes equations (RANSE) aimed at predicting turbulent single- and two-phase flows around ship hulls. The CFD capability employs a finite-volume discretization that allows use of arbitrary polyhedral elements. The free surface is captured using a volume-fraction method capable of accurately resolving sharp interfaces. NavyFOAM has been developed using an open-source CFD software tool-kit (OpenFOAM) that draws heavily upon object-oriented programming. The numerical methods and the physical models in the original version of OpenFOAM have been upgraded in an effort to improve accuracy and robustness of numerical solutions. The details of NavyFOAM V1.0 including the numerical methods and the physical models are described in this report. NavyFOAM V1.0 is demonstrated for a number of flows including: underwater bodies, turbulent free surface flows around the DTMB 5415 model and the KVLCC2 double-model. It is shown that the RANSE based approach can predict, with good accuracy, most of the salient features of the turbulent free-surface flows around the subject hulls including resistance, wave elevation, hull boundary layer and wake.

Administrative Information

The work described in this report was performed by the Computational Hydromechanics Division (Code 5700) of the Hydromechanics Department at the Naval Surface Warfare Center, Carderock Division (NSWCCD). This effort has been funded by the Department of Defense High Performance Computing Modernization Program (HPCMP) under the Computational Research and Engineering Acquisition Tools and Environments (CREATE) Ship's Hydrodynamics Project.

Introduction

In the past two decades, computational fluid dynamics (CFD) has been established as an indispensible tool for design and analysis in ship hydrodynamics. CFD has also significantly expanded its realm, covering a broad spectrum of applications including: resistance, powering, propulsion, maneuvering and seakeeping. The geometrical, physical and operational complexity involved in ship hydrodynamics applications has led to the addition of many features and functionalities in CFD codes. Furthermore, CFD is frequently called upon to tackle multi-disciplinary applications such as fluid-structure interaction and hydroacoustics applications that require coupling of CFD codes with other computational mechanics software. Thus, general-purpose CFD codes, in attempts to cater to these diverse needs, have become increasingly larger and more complex. Software complexity is a serious issue which many legacy CFD codes face today, negatively impacting their overall efficacy in terms of quality assurance, packaging, maintenance and extensions.

The Department of Defense High Performance Computing Modernization Program (HPCMP) office, under the CREATE Ship's Hydrodynamics Project, has initiated an effort to develop a CFD capability aimed at high-fidelity, high-performance, predictions of hydrodynamic phenomena occurring around surface ships and submarines. The ultimate goal of the project is to develop a high-fidelity CFD capability that can drastically shorten the design

eycles of surfaee ships and submarines, by answering teehnieal questions on various aspects of hydrodynamic performance of naval vessels at early design stages.

To meet the top-level requirements of the program, it was considered imperative that the new CFD software be developed using modern software engineering praetiees. Among others, it was concluded that object-oriented programming (OOP) with properly designed data strueture and code architecture is essential to facilitate development, quality assuranee (QA), deployment (paekaging/release), maintenance, and extension of the software. Thus, we started with OpenFOAM (Wellcr et al.[1]), an open-source CFD software tool-kit writtcn in C++ drawing heavily upon object-oriented programming (OOP). Efforts to develop a computational framework using OpenFOAM had started out earlier with propulsors the target applications (Kim et al.[2]). The CREATE efforts have greatly benefited from our earlier works on turbulence modeling, discretization schemes and solution algorithms. As of today, the OpenFOAM-based eomputational framework eomprises a suite of modified and newly written application (top-level) solvers for single- and multi-phase flows, utilities and physies libraries built around the OpenFOAM CFD tool-kit. We loosely refer to the computational framework as "NavyFOAM" in order to distinguish it from the standard OpenFOAM offering.

NavyFOAM includes several top-level solvers, Table 1, aimed at ship hydrodynamics applieations, sRansFOAM (single-phase, steady RANSE solver), ransFSFOAM (RANSE-based free-surface solver), and ransFSDyMFOAM (RANSE-based free-surfaee solver with moving/deforming mesh), to name a few. That one has to deal with a number of top-level solvers for different applieations often surprises those who are used to the idea of developing a monolithic CFD solver that can do everything. The philosophy adopted in OpenFOAM eschews the monolithic approach.

This report consists of a number of sections, including:

- Teehnieal deseription
- User's Guide
- Example Results
- Utility Programs
- Tutorials

Technical Description gives an overview of the theoretical formulation and the numerical methods used in the RANSE solvers in NavyFOAM.

User's Guide is intended to help users learn how to run the eodes without delving into the details of the implementations. This chapter should be considered as an annex to OpenFOAM's User's Guide. Those who are interested only in running the top-level solvers provided in NavyFOAM should read this ehapter and the **Tutorials** and ean skip the other ehapters if they want to.

Example Results presents example problems run with NavyFOAM seleeted from various applieations ineluding surface ships and underwater bodies.

Utility Programs is an appendix that describes the top-level applieations newly added to facilitate post-processing of the CFD results obtained using NavyFOAM.

Tutorials given in the appendices provide step-by-step instructions starting from setting up the case to running the NavyFOAM solvers to exporting the results for post-processing.

Table 1. Examples of top-level RANS solvers built using the OpenFOAM toolkit for marine propulsor applications (GGI: grid-to-grid interpolation)

Solver	Features/Functionalities	Applications
sRansFoam	Single-phase, steady, RANSE, flow solver in the inertial frame	Underwater bodies (without free-surface effects)
ransFSFoam	Two-phase, unsteady, RANSE, flow solver in the inertial or rotating frame with GGI	Surface ships with fixed sinkage and trim Propellers in open water with uniform inflow
ransFSDyMFoam	Two-phase, unsteady RANSE, flow solver in the inertial frame with dynamic mesh motion with GGI	Surface ships with dynamic sinkage and trim prediction Propellers with non-uniform inflow

Technical Description

NavyFOAM employs a cell-centered finite-volume method based on a multi-dimensional linear reconstruction scheme that permits use of arbitrary polyhedral elements including quadrilateral, hexahedral, triangular, tetrahedral, pyramidal, prismatic, and hybrid meshes. The solution gradients at cell centers can be evaluated by applying the Green-Gauss theorem or by the least-square method. Spatial and temporal discretizations formally have up to second-order accuracy. The volume-fraction equation is solved using an implicit solver. The discretized governing equations can be solved using a choice of iterative linear solvers such as point-implicit Gauss-Seidel or algebraic multi-grid (AMG) methods. Velocity coupling to ensure mass conservation (continuity) is effected using a projection algorithm. The entire NavyFOAM solver suite can be run in parallel using domain decomposition and a public version of MPI (OpenMPI) for message passing.

Governing Equations

The governing equations adopted in NavyFOAM consist of the continuity (mass conservation) equation, momentum equations, turbulent transport equations, and a volume-fraction equation. Which equations are solved in a top-level solver depends on whether the flow is single-phase or multiphase.

Single Phase Flow

For single phase incompressible flow, the governing equations consist of the continuity equation, the momentum equation, and the turbulence transport equation(s). The continuity equation can be written in a differential form as:

$$\nabla \cdot \bar{V} = 0 \tag{1}$$

The momentum equation can be written as:

$$\frac{\partial \bar{V}}{\partial t} + \nabla \cdot (\bar{V}\bar{V}) = -\nabla P + \nabla \cdot \left\{ \nu_{eff} \left(\nabla \bar{V} + \nabla \bar{V}^T \right) \right\} \tag{2}$$

where $\bar{V}$ is the velocity vector, $P \equiv \frac{p}{\rho}$ is the modified pressure, p is the hydrodynamic pressure, ρ is the density, $\nu_{eff} \equiv \nu + \nu_t$ is the effective viscosity, ν is the kinematic viscosity, and ν_t is the turbulent eddy viscosity.

Multiphase Flow

In the volume of fluid (VOF) method, the governing equations for two-phase flow consist of the continuity equation, the momentum equation, the convection equation for volume fraction, and the turbulence transport equation(s). The continuity equation is given by

$$\nabla \cdot \bar{V} = 0 \tag{3}$$

The momentum equation is given by

$$\frac{\partial(\rho\bar{V})}{\partial t}+\nabla\cdot(\rho\bar{V}\bar{V})=-\nabla p+\nabla\cdot\left\{\mu_{eff}\left(\nabla\bar{V}+\nabla\bar{V}^{T}\right)\right\}+\rho\bar{g}+\sigma\kappa\nabla\gamma \tag{4}$$

where γ is the volume fraction, $\bar{g}$ is the gravitational acceleration vector, σ is the surface tension coefficient, and κ is the interface curvature, $\mu_{eff}\equiv\mu+\mu_t$ is the effective viscosity, μ is the dynamic viscosity, and μ_t is the turbulent eddy viscosity. The density is calculated by $\rho=\gamma\rho_1+(1-\gamma)\rho_2$, and the dynamic viscosity by $\mu=\mu(\mu_1,\mu_2,\gamma)$. The subscripts "1" and "2" refer to the two phases or fluids. The convective transport equation for volume-fraction is

$$\frac{\partial\gamma}{\partial t}+\nabla\cdot(\bar{V}\gamma)=0 \tag{5}$$

Turbulence Models

NavyFOAM allows users to choose from the entire turbulence model suite available in OpenFOAM. NavyFOAM additionally offers a Wilcox's k-ω turbulence model (Wilcox[3]), a modified SST k-ω model, and a custom version of Spalart and Allmaras' one-equation model. Wall models are implemented in these newly available turbulence models so that the models can be used with either a wall-resolving (y+ < 1) or a wall-skipping (y+ >30) mesh.

Spatial and Temporal Discretization

Gradient Schemes

Gauss Integration. The gradient can be calculated using the Gauss theorem

$$\int_{\Omega_e}\nabla\varphi d\Omega=\int_{\Gamma_e}\bar{n}\varphi d\Gamma \tag{6}$$

Assuming the gradient is constant in a cell, (6) can be approximated as

$$\nabla\varphi_P\approx\frac{1}{|\Omega_e|}\sum_f\varphi_f\bar{S}_f \tag{7}$$

where the subscript P denotes the cell center, $|\Omega_e|$ is the volume of the cell, and $\bar{S}_f$ is the area vector of each face of the cell.

Cell-Based Calculation. In a cell-based approach, the face-center value φ_f in (7) is calculated using the cell-center value of φ in neighboring cells, as shown in Figure 1. This approach is used in OpenFOAM.

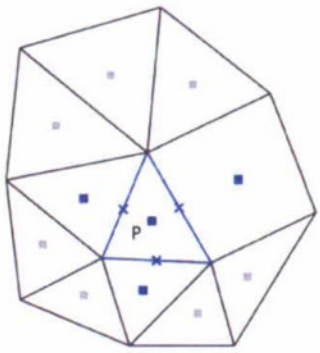


Figure 1. Stencil in cell-based gradient calculation in two-dimensions

Node-Based Calculation In a node-based approach, the nodal value of φ (red circles in Figure 2) is first calculated using all neighboring cells of the node, then the face-center value φ_f in (7) is calculated using nodal values. As a result, the stencil involved in the gradient calculation (all blue squares in Figure 2) is much larger than that in the cell-based approach.

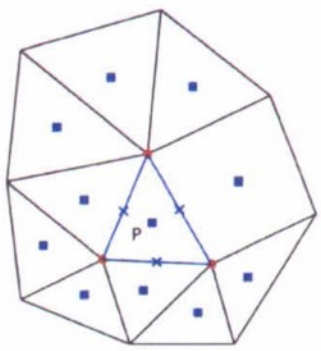


Figure 2. Stencil in node-based gradient calculation in two-dimensions

Two types of node-based gradient calculations, i.e. the *Pseudo-Laplacian-Weighted-Averaging* (PLWA) and the *Inverse-Distance-Weighted-Averaging* (IDWA) have been implemented in NavyFOAM. They differ in the way that the cell-centered volume field is interpolated to the node point field. More details are described later.

Least Squares. The concept of least-squares calculation of gradients is easily illustrated in two-dimensions. There should be no difficulty to extend it to three-dimensions. Suppose we want to calculate the gradient of φ at the center of cell i, see Figure 3, the neighboring cells are k =1, 2, and 3.

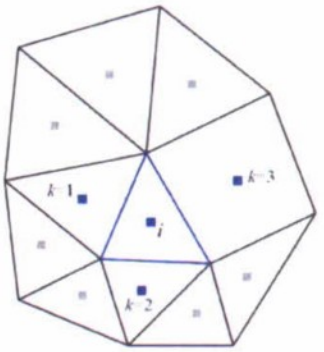


Figure 3. Stencil in least-squares gradient calculation in two-dimensions

In a general form, let N_b be the total number of neighboring cells, the neighboring cell-center value of φ can be written as a Taylor expansion about the center of cell i.

$$\varphi_k = \varphi_i + (\nabla\varphi)_i \cdot \Delta\vec{r}_{ik} + \varepsilon_{ik} \quad \text{for } k = 1, \ldots, N_b \tag{8}$$

where ε_{ik} represents the higher-order errors. Defining a total error as the sum of weighted errors using

$$E = \sum_{k=1}^{N_b} \varepsilon_{ik}^2 \tag{9}$$

where w_{ik} is the weighting factor. Omitting the index i for brevity, Equation (8) can be written as

$$\varepsilon_k = \varphi_k - \varphi - \nabla\varphi \cdot \Delta\vec{r}_k \quad \text{for } k = 1, \ldots, N_b$$

or in the form of Cartesian components

$$\varepsilon_k = \varphi_k - \varphi - \begin{bmatrix} \varphi_x \\ \varphi_y \\ \varphi_z \end{bmatrix} \cdot \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta z_k \end{bmatrix} \quad \text{for } k = 1, \ldots, N_b \tag{10}$$

Substituting (10) into (9) and setting

$$\frac{\partial E}{\partial \varphi_x} = 0, \quad \frac{\partial E}{\partial \varphi_y} = 0, \quad \textit{and} \quad \frac{\partial E}{\partial \varphi_z} = 0$$

to minimize the total error, one has

$$\begin{bmatrix} \sum_{k=1}^{N_b} w_k(\Delta x_k)^2 & \sum_{k=1}^{N_b} w_k(\Delta x_k \Delta y_k) & \sum_{k=1}^{N_b} w_k(\Delta x_k \Delta z_k) \\ \sum_{k=1}^{N_b} w_k(\Delta x_k \Delta y_k) & \sum_{k=1}^{N_b} w_k(\Delta y_k)^2 & \sum_{k=1}^{N_b} w_k(\Delta y_k \Delta z_k) \\ \sum_{k=1}^{N_b} w_k(\Delta x_k \Delta z_k) & \sum_{k=1}^{N_b} w_k(\Delta y_k \Delta z_k) & \sum_{k=1}^{N_b} w_k(\Delta z_k)^2 \end{bmatrix} \begin{bmatrix} \varphi_x \\ \varphi_y \\ \varphi_z \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{N_b} w_k \Delta x_k(\varphi_k - \varphi) \\ \sum_{k=1}^{N_b} w_k \Delta y_k(\varphi_k - \varphi) \\ \sum_{k=1}^{N_b} w_k \Delta z_k(\varphi_k - \varphi) \end{bmatrix} \tag{11}$$

The solution to the liner system in (11) gives the gradient ealculated in the least-squares sense.

Convection Schemes

In finite volume methods, the eonveetion term ean be ealculated using thc Gauss theorem

$$\int_{\Omega_e} \nabla \cdot (\vec{V}\varphi) d\Omega = \int_{\Gamma_e} \vec{n} \cdot (\vec{V}\varphi) d\Gamma \tag{12}$$

The surfaec integration in (12) can be approximately calculated as

$$\int_{\Gamma_e} \vec{n} \cdot (\vec{V}\varphi) d\Gamma \approx \sum_f (\vec{V} \cdot \vec{S})_f \varphi_f \tag{13}$$

The eonvection scheme dctermincs how thc face-centcr value φ_f is ealculatcd. The most widely rcferenced boundedness criterion using the normalizcd variable approaeh is Gaskell and Lau's Conveetion-Boundedness Criterion (CBC) (Gaskell & Lau[4]).

Convection Boundedness Criterion (CBC). The concept of CBC is easily illustratcd in one-dimension as shown in Figure 4, where D is the donor eell, U is the upstream eell, and A is the aeceptor eell.

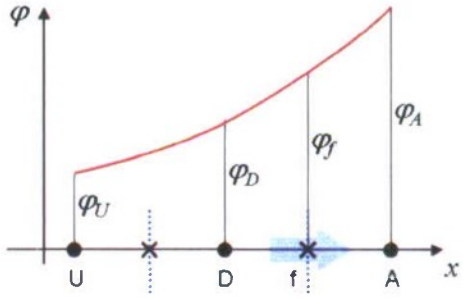


Figure 4. Illustration of nodes and faee in onc-dimension

Defining a normalizcd variable

$$\tilde{\varphi} = \frac{\varphi - \varphi_U}{\varphi_A - \varphi_U},$$

wc thus have

$$\tilde{\varphi}_D = \frac{\varphi_D - \varphi_U}{\varphi_A - \varphi_U} \quad \text{and} \quad \tilde{\varphi}_f = \frac{\varphi_f - \varphi_U}{\varphi_A - \varphi_U}.$$

Based on the normalized variables, CBC states the following local boundedness criterion

$$\begin{aligned} &\tilde{\varphi}_D \le \tilde{\varphi}_f \le 1 \quad \text{for} \quad \tilde{\varphi}_D \in [0,1] \\ &\tilde{\varphi}_f = \tilde{\varphi}_D \quad \text{for} \quad \tilde{\varphi}_D \notin [0,1] \end{aligned} \tag{14}$$

The graphical representation of CBC is often shown in the *normalized variable diagram* (NVD) (Leonard[5]) of Figure 5.

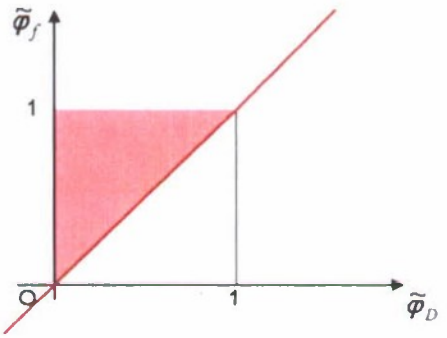


Figure 5. Normalized variable diagram (NVD) with CBC (the shaded region)

In general, the normalized face-center value can be written as a funetion of the normalized value of the donor cell

$$\tilde{\varphi}_f = f(\tilde{\varphi}_D) \tag{15}$$

Linear Schemes. If the funetion f in (15) is linear, the seheme is called a linear scheme. Examples of linear schemes include:

Central differencing (CD) scheme

$$\tilde{\varphi}_f = \frac{1}{2}\tilde{\varphi}_D + \frac{1}{2} \tag{16}$$

Upwind differencing (UD) scheme

$$\tilde{\varphi}_f = \tilde{\varphi}_D \tag{17}$$

Downwind differencing (DD) scheme

$$\tilde{\varphi}_f = 1 \tag{18}$$

Quadratic Upstream Interpolation for Convective Kinetics (QUICK) scheme

$$\tilde{\varphi}_f = \frac{3}{4}\tilde{\varphi}_D + \frac{3}{8} \tag{19}$$

Warming & Beam Second-Order Upwind (SOUD) scheme

$$\tilde{\varphi}_f = \frac{5}{4}\tilde{\varphi}_D + \frac{1}{8} \tag{20}$$

The NVD of these schemes and the CBC region are shown in Figure 6.

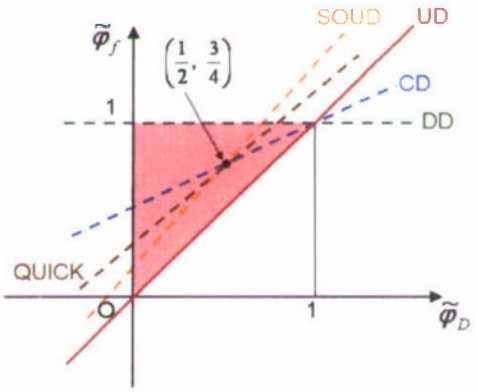


Figure 6. NVD of linear schemes with CBC

Nonlinear Schemes. The order barrier (Godunov[6]) for linear schemes implies that the CBC schemes with accuracy of second-order or above must be nonlinear schemes, i.e. the function f in (15) must be nonlinear. Hereafter, we summarize some of the CBC schemes (limited schemes in NavyFOAM).

vanLeer Scheme (van Leer[7])

$$f(\tilde{\varphi}) = \begin{cases} 2\tilde{\varphi} - \tilde{\varphi}^2, & \tilde{\varphi} \in [0,1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0,1] \end{cases} \tag{21}$$

The NVD of the scheme and the CBC region are shown in Figure 7.

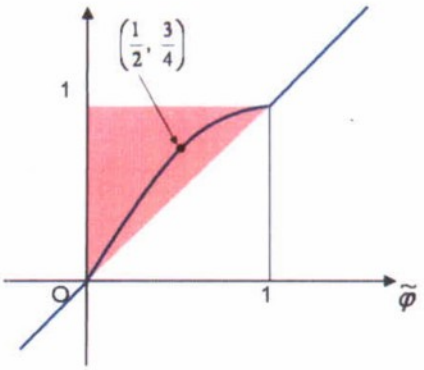


Figure 7. NVD of vanLeer scheme with CBC

Gamma Scheme (Jasak et al.[8])

$$f(\tilde{\varphi}) = \begin{cases} -\frac{1}{2\beta}\tilde{\varphi}^2 - (1+\frac{1}{2\beta})\tilde{\varphi}, & \tilde{\varphi} \in [0, \beta] \\ \frac{1}{2}\tilde{\varphi} + \frac{1}{2} & \tilde{\varphi} \in [\beta, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases} \quad \text{with } \frac{1}{10} \le \beta \le \frac{1}{2} \tag{22}$$

The NVD of the scheme and the CBC region are shown in Figure 8.

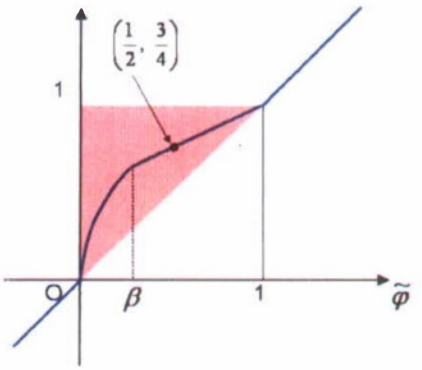


Figure 8. NVD of Gamma scheme with CBC

HYPER-C Scheme (Leonard[9])

$$f(\tilde{\varphi}) = \begin{cases} \min(1, \frac{1}{C_f}\tilde{\varphi}), & \tilde{\varphi} \in [0, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases} \tag{23}$$

The HYPER-C scheme requires that the local Courant number $C_f \le 1$. The NVD of the scheme and the CBC region are shown in Figure 9.

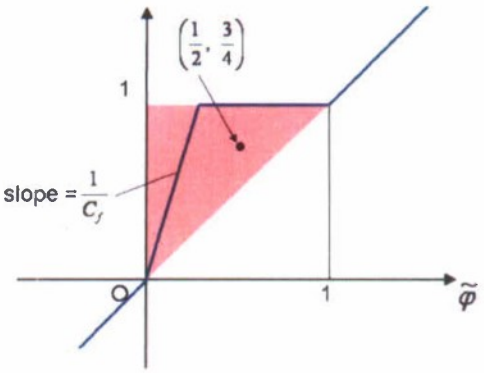


Figure 9. NVD of HYPER-C scheme with CBC

Ultimate-Quickest (UQ) Scheme (Leonard[9])

$$f(\tilde{\varphi}) = \begin{cases} \min\{\frac{8C_f\tilde{\varphi}+(1-C_f)(6\tilde{\varphi}+3)}{8}, f_{\text{HYPER\_C}}(\tilde{\varphi})\}, & \tilde{\varphi} \in [0,1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0,1] \end{cases} \tag{24}$$

The scheme requires that the local Courant number $C_f \leq 1$. The NVD of the scheme and the CBC region are shown in Figure 10.

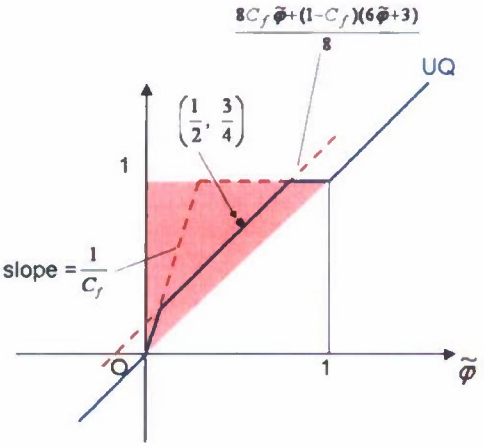


Figure 10. NVD of UQ scheme with CBC

Compressive Interface Capturing Scheme for Arbitrary Meshes (CICSAM) (Ubbink& Issa[10])

$$f(\tilde{\varphi}) = \gamma_f f_{\text{HYPER-C}}(\tilde{\varphi}) + (1-\gamma_f) f_{\text{UQ}}(\tilde{\varphi}) \tag{25}$$

where $\gamma_f = \min\left(\frac{1+\cos 2\theta_f}{2}, \ 1\right)$. θ_f is the angle between the normal unit vector of the front (interface between two phases) and the vector pointing from D to A, see Figure 11.

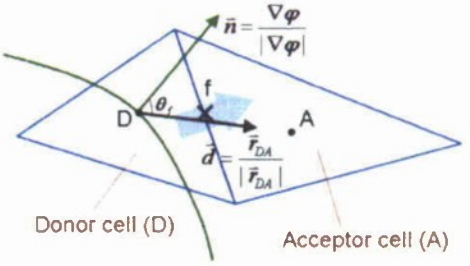


Figure 11. Angle θ_f

The NVD of the scheme and the CBC region are shown in Figure 12.

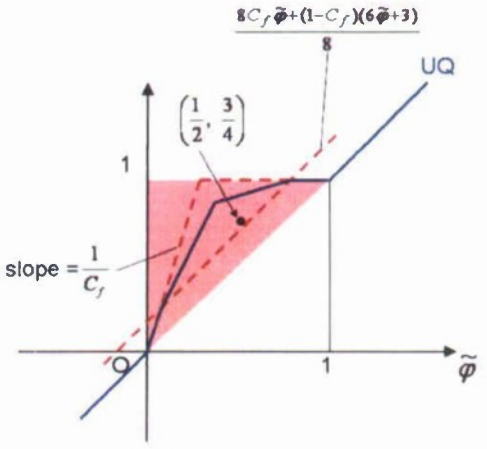


Figure 12. NVD of ClCSAM scheme with CBC

High Resolution Interface Capturing Scheme (HRIC) (Muzaferija & Peric[11]) Let

$$f_1(\tilde{\varphi})=\begin{cases}2\tilde{\varphi}, & \tilde{\varphi}\in[0,\frac{1}{2}]\\ 1, & \tilde{\varphi}\in[\frac{1}{2},1]\\ \tilde{\varphi}, & \tilde{\varphi}\notin[0,1]\end{cases}$$ and

$$f_2(\tilde{\varphi})=\gamma_f f_1(\tilde{\varphi})+(1-\gamma_f)\tilde{\varphi} \quad \text{with } \gamma_f=\sqrt{|\cos\theta_f|},$$

The HRIC scheme can be written as

$$f(\tilde{\varphi})=\begin{cases}f_2(\tilde{\varphi}), & C_f<0.3\\ \frac{0.7-C_f}{0.4}f_2(\tilde{\varphi})+\left(1-\frac{0.7-C_f}{0.4}\right)\tilde{\varphi}, & 0.3\le C_f\le 0.7\\ \tilde{\varphi}, & C_f>0.7\end{cases} \tag{26}$$

The NVD of the scheme and the CBC region are shown in Figure 13.

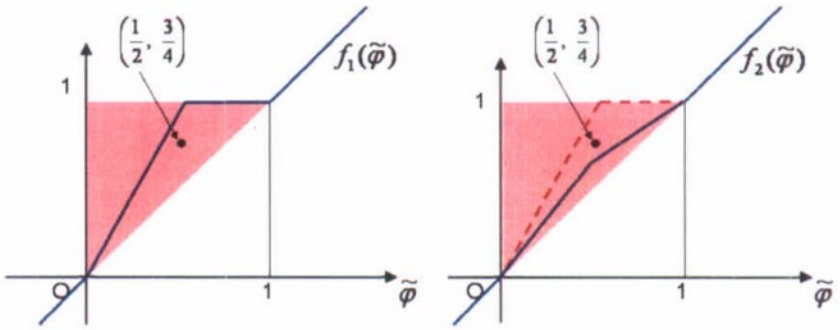


Figure 13. NVD of HRIC scheme with CBC

Modified HRIC (MHRIC) (Park et al.[12]) Let

$$f_1(\tilde{\varphi}) = \begin{cases} 2\tilde{\varphi}, & \tilde{\varphi} \in [0, \frac{1}{2}] \\ 1, & \tilde{\varphi} \in [\frac{1}{2}, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases}$$

and

$$f_2(\tilde{\varphi}) = \begin{cases} \min\left(\frac{6\tilde{\varphi}+3}{8}, \ f_1(\tilde{\varphi})\right), & \tilde{\varphi} \in [0, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases}$$

MHRIC can be written as

$$f(\tilde{\varphi}) = \gamma_f f_1(\tilde{\varphi}) + (1 - \gamma_f) f_2(\tilde{\varphi}) \tag{27}$$

with $\gamma_f = \sqrt{|\cos\theta_f|}$. The NVD of the scheme and the CBC region are shown in Figure 14.

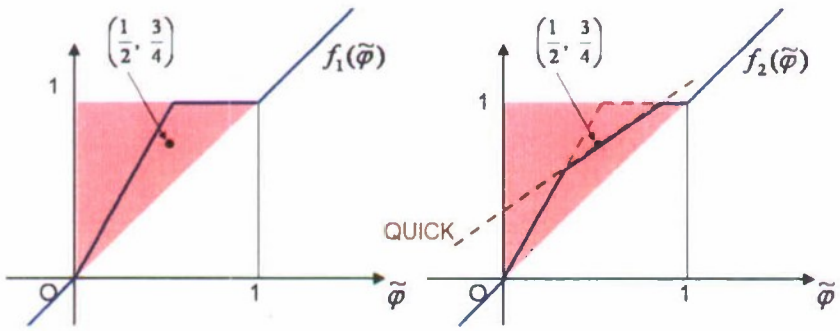


Figure 14. NVD of MHRIC scheme with CBC

Inter-Gamma Scheme (interGamma) (Jasak & Weller[13])

$$f(\tilde{\varphi}) = \begin{cases} -2\tilde{\varphi}^2 + 3\tilde{\varphi}, & \tilde{\varphi} \in [0, \frac{1}{2}] \\ 1 & \tilde{\varphi} \in [\frac{1}{2}, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases} \tag{28}$$

The NVD of the scheme and the CBC region are shown in Figure 15.

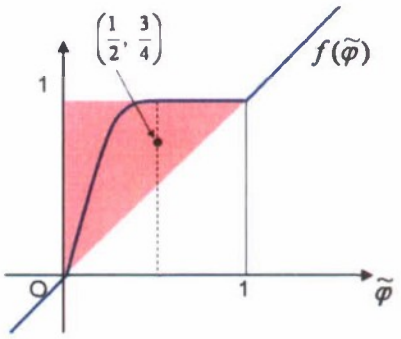


Figure 15. NVD of inter-Gamma scheme with CBC

Modified inter-Gamma Scheme (interGammaM) Let

$$f_1(\tilde{\varphi}) = \begin{cases} -2\tilde{\varphi}^2 + 3\tilde{\varphi}, & \tilde{\varphi} \in [0, \frac{1}{2}] \\ 1 & \tilde{\varphi} \in [\frac{1}{2}, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases}$$

be the original inter-Gamma scheme, which can be modified as follows

$$f(\tilde{\varphi}) = \begin{cases} f_1(\tilde{\varphi}), & C_f < 0.3 \\ \frac{0.7-C_f}{0.4} f_1(\tilde{\varphi}) + \left(1 - \frac{0.7-C_f}{0.4}\right)\tilde{\varphi}, & 0.3 \le C_f \le 0.7 \\ \tilde{\varphi}, & C_f > 0.7 \end{cases} \tag{29}$$

The NVD of the scheme and the CBC region are shown in Figure 16.

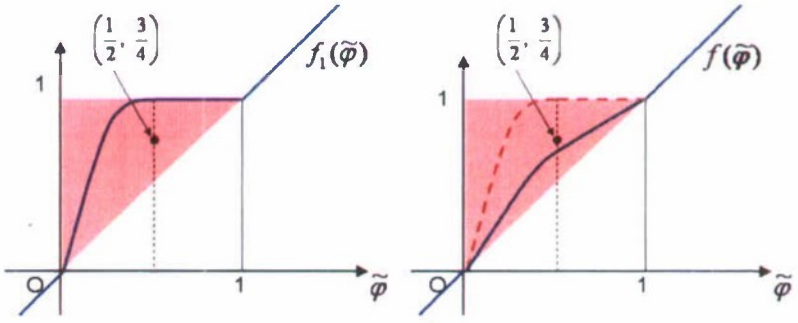


Figure 16. NVD of interGammaM scheme with CBC

Modified inter-Gamma Scheme (interGammaMD) Let

$$f_1(\tilde{\varphi}) = \begin{cases} -2\tilde{\varphi}^2 + 3\tilde{\varphi}, & \tilde{\varphi} \in [0, \frac{1}{2}] \\ 1 & \tilde{\varphi} \in [\frac{1}{2}, 1] \\ \tilde{\varphi}, & \tilde{\varphi} \notin [0, 1] \end{cases}$$

be the original inter-Gamma scheme, and

$$f_2(\tilde{\varphi}) = \gamma_f f_1(\tilde{\varphi}) + (1 - \gamma_f)\tilde{\varphi} \quad \text{with } \gamma_f = \sqrt{|\cos\theta_f|} \tag{30}$$

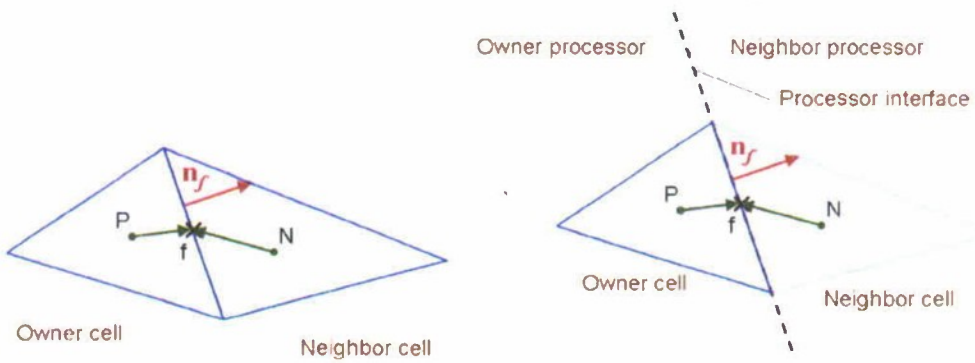
The interGammaMD scheme can be written as

$$f(\tilde{\varphi}) = \begin{cases} f_2(\tilde{\varphi}), & C_f < 0.3 \\ \frac{0.7 - C_f}{0.4} f_2(\tilde{\varphi}) + \left(1 - \frac{0.7 - C_f}{0.4}\right)\tilde{\varphi}, & 0.3 \le C_f \le 0.7 \\ \tilde{\varphi}, & C_f > 0.7 \end{cases} \tag{31}$$

Interpolation Schemes

Cell-based Surface Interpolation Schemes.

In cell-center-based finite volume methods, it is often required to calculate the face-center value. The interpolation schemes needed for non-convection terms will be introduced in this section. The situation is illustrated in Figure 17. The owner and neighbor cells of the face may or may not be located within a mesh block of a single processor.



(a) internal face (b) coupled processor face

Figure 17. Cell-based interpolation of face-center value

Linear Interpolation Surface Interpolation Scheme. The linear interpolation calculates the face-center value as

$$\varphi_f = \lambda\varphi_P + (1 - \lambda)\varphi_N \tag{32}$$

where λ is the weighting factor calculated as

$$\lambda = \frac{|\vec{n}_f \cdot \Delta\vec{r}_{Nf}|}{|\vec{n}_f \cdot \Delta\vec{r}_{Pf}| + |\vec{n}_f \cdot \Delta\vec{r}_{Nf}|}$$

The linear interpolation scheme may produce large errors due to non-orthogonality and skewness of unstructured meshes.

reconCentral Surface Interpolation Scheme. The face-center value can be reconstructed using both the value and the gradient at neighboring cell centers.

$$\varphi_f = \frac{1}{2}\left[\varphi_P + (\nabla\varphi)_P \cdot \vec{r}_{Pf} \;+\; \varphi_N + (\nabla\varphi)_N \cdot \vec{r}_{Nf}\right] \tag{33}$$

Upwind Deferred Correction (UPDC) Surface Interpolation Schemes. To improve stability, the face-center value can sometimes be calculated using the concept of deferred correction (Khosla & Rubin[14]; Hayase et al.[15]).

$$\varphi_f = \varphi_f^{\mathrm{FOU}} + (\varphi_f^{\mathrm{H}} - \varphi_f^{\mathrm{FOU}})^{\mathrm{old}} \tag{34}$$

where φ_f^{FOU} is the value calculated using a first-order upwind scheme, and φ_f^{H} is the value calculated using higher-order interpolation schemes. The superscript "old" represents the previous time or iteration step. The candidates for the higher-order scheme may include the reconCentral scheme and some of the higher-order limited schemes in the next section.

The first-order upwind scheme can be written as

$$\varphi_f^{\mathrm{FOU}} = \mathrm{sign}^+(F_f)\varphi_P + [1 - \mathrm{sign}^+(F_f)]\varphi_N \tag{35}$$

where $F_f = \vec{n}_f \cdot \vec{V}_f$ is the volume flux through the face, and

$$\mathrm{sign}^+(F_f) = \begin{cases} 1, & F_f \geq 0 \\ 0, & F_f < 0 \end{cases}$$

The higher-order normalized variable based limited schemes can be written as

$$\varphi_f^{\mathrm{H}} = \varphi_P + \frac{1}{2}\Psi(r)(\varphi_N - \varphi_P) \tag{36}$$

where $\Psi(r)$ is the limiter function. Substituting (35) and (36) into (34) yields

$$\varphi_f = \begin{cases} \varphi_P + [\frac{1}{2}\Psi(r)(\varphi_N - \varphi_P)]^{\mathrm{old}}, & F_f \geq 0 \\ \varphi_N + [\frac{1}{2}\Psi(r)(\varphi_P - \varphi_N)]^{\mathrm{old}}, & F_f < 0 \end{cases} \tag{37}$$

Node-Based Surface Interpolation Schemes.

We consider a face of any polyhedral cell

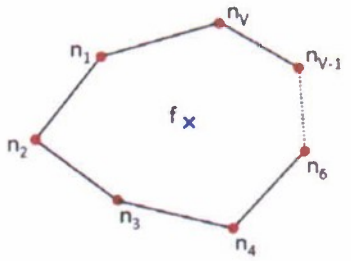


Figure 18. Node-based interpolation of face-center value

Let N_v be the total number of vertices of faces, the face-center value is calculated as the average of the nodal values

$$\varphi_f = \frac{1}{N_v}\sum_{k=1}^{N_v}\varphi_{n_k} \tag{38}$$

The nodal values are calculated using volume-to-point interpolation described in the following section.

Volume-to-Point Interpolation Schemes

It is easier to illustrate in two-dimensions and the extension to three-dimension is rather straightforward.

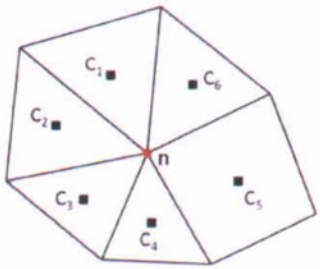


Figure 19. Stencil in cell-based gradient calculation in two-dimensions

In volume-to-point interpolation, the nodal value φ_n is calculated as a weighted averaging of surrounding cell-center values

$$\varphi_n = \sum_{i=1}^{N_n}(\varphi_{c,i}w_{c,i} / \sum_{j=1}^{N_n} w_{c,j}) \tag{39}$$

where N_n is the total number of neighboring cells of node n, $w_{c,i}$ is the weighting factor.

Volume-to-Point Interpolation Based on PLWA. In *Pseudo-Laplacian-Weighted-Averaging* (PLWA) (Holmes & Connell[16]; Frink[17]; Kim et al.[18]), the weighting factors in Equation (39) are calculated by solving the following optimization problem.

Giving the constraint

$$\begin{aligned} L(x_n) &= \sum_{i=1}^{N_n} w_{c,i}(x_{c,i} - x_n) = 0 \\ L(y_n) &= \sum_{i=1}^{N_n} w_{c,i}(y_{c,i} - y_n) = 0 \\ L(z_n) &= \sum_{i=1}^{N_n} w_{c,i}(z_{c,i} - z_n) = 0 \end{aligned} \quad , \tag{40}$$

we need to find the weighting factors $w_{c,i}$ that minimize the cost function

$$C = \sum_{i=1}^{N_n} (r_{c,i}\Delta w_{c,i})^2 \tag{41}$$

where $\vec{r}_{c,i} = (x_{c,i}, y_{c,i}, z_{c,i})$ is the position vector of the cell center, $\vec{r}_n = (x_n, y_n, z_n)$ is the position vector of the node n, and $r_{c,i} = |\vec{r}_{c,i} - \vec{r}_n| = \sqrt{(x_{c,i} - x_n)^2 + (y_{c,i} - y_n)^2 + (z_{c,i} - z_n)^2}$. The $\Delta w_{c,i}$ is related to the weighting factor by $w_{c,i} = 1 + \Delta w_{c,i}$.

Using the method of Lagrange multipliers, the Lagrange function is defined as

$$\Lambda(w_{c,1}, w_{c,2}, \ldots, w_{c,N_n};\ \lambda_x, \lambda_y, \lambda_z) = C - 2[\lambda_x L(x_n) + \lambda_y L(y_n) + \lambda_z L(z_n)] \tag{42}$$

The optimal solution is found by solving the following equation

$$\nabla_{w_{c,1}, w_{c,2}, \ldots, w_{c,N_n};\ \lambda_x, \lambda_y, \lambda_z} \Lambda(w_{c,1}, w_{c,2}, \ldots, w_{c,N_n};\ \lambda_x, \lambda_y, \lambda_z) = 0 \tag{43}$$

which can be written in matrix form

$$\begin{bmatrix} [\mathrm{diag}(r^2)] & -[\Delta \mathbf{r}] \\ [\Delta \mathbf{r}]^T & [\mathbf{0}] \end{bmatrix} \begin{bmatrix} \Delta \mathbf{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -\mathbf{R} \end{bmatrix} \tag{44}$$

where

$$[\mathrm{diag}(r^2)] = \begin{bmatrix} r_{c,1}^2 & & & \\ & r_{c,2}^2 & & \\ & & \ddots & \\ & & & r_{c,N_n}^2 \end{bmatrix},$$

$$[\Delta\mathbf{r}] = \begin{bmatrix} x_{c,1} - x_n & y_{c,1} - y_n & z_{c,1} - z_n \\ x_{c,2} - x_n & y_{c,2} - y_n & z_{c,2} - z_n \\ \vdots & \vdots & \vdots \\ x_{c,N_n} - x_n & y_{c,N_n} - y_n & z_{c,N_n} - z_n \end{bmatrix},$$

$$\Delta\mathbf{w} = \begin{bmatrix} \Delta w_{c,1} \\ \Delta w_{c,2} \\ \vdots \\ \Delta w_{c,N_n} \end{bmatrix},$$

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{bmatrix}, \text{ and}$$

$$\mathbf{R} = \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N_n} (x_{c,i} - x_n) \\ \sum_{i=1}^{N_n} (y_{c,i} - y_n) \\ \sum_{i=1}^{N_n} (z_{c,i} - z_n) \end{bmatrix}.$$

Solving the linear system of equations we obtain

$$\begin{aligned} &\boldsymbol{\lambda} = -[\mathbf{I}]^{-1}\mathbf{R} \quad \text{and} \\ &\Delta\mathbf{w} = [\text{diag}(r^2)]^{-1}[\Delta\mathbf{r}]\boldsymbol{\lambda} \end{aligned} \tag{45}$$

where

$$[\mathbf{I}] = [\Delta\mathbf{r}]^{\mathrm{T}}[\text{diag}(r^2)]^{-1}[\Delta\mathbf{r}] .$$

Volume-to-Point Interpolation Based on IDWA. In *Inverse-Distance-Weighted-Averaging* (IDWA), the weighting factors in Equation (39) are calculated based on the distance between the node and each neighboring cell center.

$$w_{c,i} = \frac{1}{r_{c,i}^2} \tag{46}$$

with

$$r_{c,i} = |\vec{r}_{c,i} - \vec{r}_n| = \sqrt{(x_{c,i} - x_n)^2 + (y_{c,i} - y_n)^2 + (z_{c,i} - z_n)^2} .$$

Diffusion Schemes

In finitc volume methods, the diffusion term can be calculated using the Gauss theorem

$$\int_{\Omega_e} \nabla \cdot (\gamma \nabla \varphi) d\Omega = \int_{\Gamma_e} \vec{n} \cdot (\gamma \nabla \varphi) d\Gamma \tag{47}$$

The surface integration in (47) can be approximately calculated as

$$\int_{\Gamma_e} \vec{n} \cdot (\gamma \nabla \varphi) d\Gamma \approx \sum_f (\gamma \vec{n} \cdot \nabla \varphi)_f S_f = \sum_f (\gamma_f S_f) \left(\frac{\partial \varphi}{\partial n} \right)_f \tag{48}$$

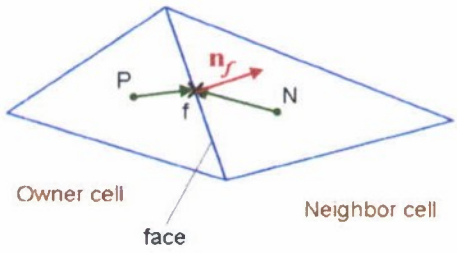


Figure 20. Calculating face-normal gradient

Thc face-normal gradient can be calculated as

$$\left(\frac{\partial \varphi}{\partial n} \right)_f = \frac{\varphi_N - \varphi_P}{|\Delta \vec{r}_{PN}|} + \underbrace{\vec{C}_f \cdot \overline{(\nabla \varphi)}_f}_{\text{snGrad correction}} \tag{49}$$

where $\vec{C}_f = \vec{n}_f - \Delta \vec{r}_{PN} / |\Delta \vec{r}_{PN}|$ is thc non-orthogonal correction vector, and

$$\overline{(\nabla \varphi)}_f = \lambda \nabla \varphi_P + (1 - \lambda) \nabla \varphi_N$$

λ is thc inverse-distance weighting factor.

Solution Algorithms

Pressure-Velocity Coupling

Solving the Pressure Equation. The momentum equation can be written in discretized form as a system of linear equations

$$\mathbf{A} \vec{V}^* + \tilde{\nabla} p^* = \tilde{\tilde{f}} \tag{50}$$

where $\vec{V}^*$ is the velocity vector,

p^* is the guessed pressure,

$$\widetilde{\nabla} p^* = \int_{\Omega} \nabla p^* d\Omega$$

$\mathbf{A} = \mathbf{D} - \mathbf{B} = (\mathbf{D}_{in} + \mathbf{D}_{bc}) - \mathbf{B}$ is the matrix of the linear equations,

$\mathbf{D} = (\mathbf{D}_{in} + \mathbf{D}_{bc}) = \text{diag}(\mathbf{A})$,

$\mathbf{D}_{in}$ = part of diag(**A**) contributed from internal faces,

$\mathbf{D}_{bc}$ = part of diag(**A**) contributed from boundary faces,

$-\mathbf{B} = \mathbf{A} - \mathbf{D}$ = off-diagonal part of **A**,

$\widetilde{\widetilde{f}} = \widetilde{\widetilde{f}}_{in} + \widetilde{\widetilde{f}}_{bc}$ = the source vector absorbing any explicit term and source term,

$\widetilde{\widetilde{f}}_{in}$ = part of source vector contributed from internal faces, and

$\widetilde{\widetilde{f}}_{bc}$ = part of source vector contributed from boundary faces.

Equation (50) can be written as

$$(\mathbf{D} - \mathbf{B})\vec{V}^* + \widetilde{\nabla} p^* = \widetilde{\widetilde{f}} \tag{51}$$

Giving guessed pressure, p^*, Equation (51) is solved for velocity, $\vec{V}^*$. This is the predictor step of the SIMPLE or PISO method. Because the velocity obtained from the predictor step doesn't satisfy the continuity equation, both the pressure p^* and velocity $\vec{V}^*$ need to be corrected.

In order to derive the discretized pressure equation, the following algebraic manipulation was applied to the momentum equation. The matrix **A** of discretized momentum equations can be written as

$$\mathbf{A} = \mathbf{D} + \mathbf{D}_{bc}^{cmptav} - \mathbf{D}_{bc}^{cmptav} - \mathbf{B} = (\mathbf{D}_{in} + \mathbf{D}_{bc}) + \mathbf{D}_{bc}^{cmptav} - \mathbf{D}_{bc}^{cmptav} - \mathbf{B} \tag{52}$$

with

$\mathbf{D}_{bc}^{cmptav}$ = component - average part of diag(**A**) contributed from boundary faces.

Let

$$\mathbf{D}_D = \mathbf{D}_{in} + \mathbf{D}_{bc}^{cmptav}$$

and

$$\mathbf{B}_D = (\mathbf{B} + \mathbf{D}_{bc}^{cmptav} - \mathbf{D}_{bc})$$

thus

$$\mathbf{A} = (\mathbf{D}_{in} + \mathbf{D}_{bc}^{cmptav}) - (\mathbf{B} + \mathbf{D}_{bc}^{cmptav} - \mathbf{D}_{bc}) = \mathbf{D}_D - \mathbf{B}_D$$

Now, Equation (50) can be written as

$$(\mathbf{D}_D - \mathbf{B}_D)\vec{V}^* + \widetilde{\nabla} p^* = \widetilde{\vec{f}} \tag{53}$$

Let $\vec{V}^{**}$ and p^{**} be corrected velocity and pressure, respectively. They should satisfy the discretized momentum equation (53), i.e.

$$(\mathbf{D}_D - \mathbf{B}_D)\vec{V}^{**} + \widetilde{\nabla} p^{**} = \widetilde{\vec{f}} \tag{54}$$

which can be approximated as

$$\mathbf{D}_D \vec{V}^{**} - \mathbf{B}_D \vec{V}^* + \widetilde{\nabla} p^{**} = \widetilde{\vec{f}} \tag{55}$$

Because $\mathbf{D}_D$ in (55) is a diagonal matrix, it is trivial to solve (55) to obtain

$$\vec{V}^{**} = \mathbf{D}_D^{-1}(\mathbf{B}_D \vec{V}^* + \widetilde{\vec{f}}) - \mathbf{D}_D^{-1}\widetilde{\nabla} p^{**} = \hat{\vec{V}} - \mathbf{D}_D^{-1}\widetilde{\nabla} p^{**} \tag{56}$$

where $\hat{\vec{V}} = \mathbf{D}_D^{-1}(\mathbf{B}_D \vec{V}^* + \widetilde{\vec{f}})$ is the *pseudo-velocity* vector.

Substituting (56) into the continuity equation, $\nabla \cdot \vec{V}^{**} = 0$, we obtain the following discretized Poisson equation for pressure

$$\nabla \cdot (\mathbf{D}_D^{-1}\widetilde{\nabla} p^{**}) = \nabla \cdot \hat{\vec{V}} \tag{57}$$

It is shown later that Equation (57) incorporates the idea of the Rhie-Chow momentum interpolation scheme.

In the PISO method, consecutive corrector steps may be used to correct pressure and velocity, the momentum equation that is satisfied at the (k+1)-th steps is

$$\mathbf{D}_D \vec{V}^{k+1} - \mathbf{B}_D \vec{V}^k + \widetilde{\nabla} p^{k+1} = \widetilde{\vec{f}} \tag{58}$$

thus

$$\vec{V}^{k+1} = \mathbf{D}_D^{-1}(\mathbf{B}_D \vec{V}^k + \widetilde{\vec{f}}) - \mathbf{D}_D^{-1}\widetilde{\nabla} p^{k+1} = \hat{\vec{V}} - \mathbf{D}_D^{-1}\widetilde{\nabla} p^{k+1} \tag{59}$$

where $\hat{\vec{V}} = \mathbf{D}_D^{-1}(\mathbf{B}_D \vec{V}^k + \widetilde{\vec{f}})$. Note that $k = 0$ represents the predictor step, i.e. $\vec{V}^0 = \vec{V}^*$.

Substituting (59) into the continuity equation $\nabla \cdot \vec{V}^{k+1} = 0$ yields

$$\nabla \cdot (\mathbf{D}_D^{-1}\widetilde{\nabla} p^{k+1}) = \nabla \cdot \hat{\vec{V}} \tag{60}$$

User's Guide

This section provides information for users to help with running any of the updates developed specifically as a part of NavyFOAM V1.0. In addition, there is a supplement to the original OpenFOAM User's Guide contained in Appendix A that could be useful to readers. For more detailed information on the OpenFOAM code and settings consult the OpenFOAM User's Guide: http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf. In addition, to aid users several utilities have been developed to aid in post-processing NavyFOAM results, which are described in Appendix B. Finally, tutorials have been developed that aid a user in running the codes which include: a lid driven cavity problem, a fully submerged axisymmetric body and the Wigley hull with free surface in Appendices C, D and E, respectively. It is recommended that new user's go through the tutorials to learn the preferred settings to use with the solvers.

Gradient Schemes

Navy Least Squares Gradient Schemes

The gradient scheme is specified by a sub-dictionary entry **gradSchemes** in the system finite volume dictionary file fvSchemes. Users may refer to the *OpenFOAM User Guide Ver. 1.5* Section 4.4.3 on page U-110 for more details about this sub-dictionary. Figure 21 illustrates an abbreviated example. The default gradient scheme is the one using Gauss theorem with the face-center value calculated by linear interpolation. The scheme used in calculating the gradient of pressure is **NavyLeastSquares** – where the gradient is calculated in a least squares sense, more details regarding the **NavyLeastSquares** gradient scheme can be found in the Technical Description section.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
gradSchemes
{
    //- default scheme
    default     Gauss linear;

    //- gradient scheme for pressure p
    grad(p)     NavyLeastSquares;
}
```

Figure 21. Example of **gradSchemes** sub-dictionary

Interpolation Schemes

Cell-based Surface Interpolation Schemes

reconCentral Surface Interpolation Scheme The reconCentral surface interpolation scheme calculates the face-center value of ϕ using both the value and the gradient of ϕ at the center of the owner and neighbor cells. The following example in Figure 22 shows how to specify the interpolation schemes in the sub-dictionary entry **interpolationSchemes** of the system finite volume dictionary file fvSchemes. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.1 on page U-108 for more details about this sub-dictionary. The default scheme is linear interpolation, where the face-center value is calculated as a weighted average of the value at the center of the owner and neighbor cells. The weighting factors are based on inverse distance. The **reconCentral** scheme is used to calculate the face-center value of **U**.

Dictionary file:

$(CASE_DIR)/system/fvSchemes

Sub-dictionary:

```
interpolationSchemes
{
    //- default scheme is linear interpolation
    default          linear;

    //- surface interpolation of U
    interpolate(U)   reconCentral;
}
```

Figure 22. Example of **interpolationSchemes** sub-dictionary

reconCentralDC Surface Interpolation Scheme The reconCentral is almost a second-order interpolation scheme. Besides calculating the surface interpolation at the face-center, it may also be used to improve the accuracy in discretizing the convection term in the momentum equation using deferred correction. The following example in Figure 23 shows the sub-dictionary **divSchemes** in the finite volume system file fvSchemes. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary. The convection term is integrated using the Gauss theorem and the face-center velocity **U** in the convection term takes the *Upwind Deferred Correction* (UPDC) form using the reconCentral scheme to calculate the higher-order correction. More details regarding UPDC can be found in the Technical Description Section.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in momentum equation
    div(phi, U)      Gauss reconCentralDC;
}
```

Figure 23. Example of **divSchemes** sub-dictionary

Node-based Surface Interpolation Schemes

reconPLWA Surface Interpolation Scheme The reconPLWA surface interpolation scheme calculates the face-center value as an average of the nodal values on the face. The nodal value is calculated from cell-center values using a volume-to-point interpolation scheme based on *Pseudo-Laplacian-Weighted-Averaging* (PLWA). More details of PLWA can be found in the Technical Description. The following example in Figure 24 shows the sub-dictionary entry **interpolationSchemes** in the system finite volume dictionary file fvSchemes. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.1 on page U-108 for more details about this sub-dictionary. The default interpolation scheme is linear. The next line of the dictionary shows that the **reconPLWA** scheme is used to calculate the face-center value of **U**.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
interpolationSchemes
{
    //- default scheme is linear interpolation
    default          linear;

    //- surface interpolation of U
    interpolate(U)   reconPLWA;
}
```

Figure 24. Example of **interpolationSchemes** sub-dictionary

reconIDWA Surface Interpolation Scheme The reconIDWA surface interpolation scheme calculates the face-center value as an average of the nodal value on the face. The nodal

value is calculated from cell-center values using a volume-to-point interpolation scheme based on *Inverse-Distance-Weighted-Averaging* (IDWA). More details of IDWA can be found in the Technical Description. The following example in Figure 25 shows the sub-dictionary entry **interpolationSchemes** in the system finite volume dictionary file fvSchemes. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.1 on page U-108 for more details about this sub-dictionary. The default interpolation scheme is linear. The **reconIDWA** scheme is used to calculate the face-center value of **U**.

Dictionary file:

$(CASE_DIR)/system/fvSchemes

Sub-dictionary:

```
interpolationSchemes
{
    //- default scheme is linear interpolation
    default          linear;

    //- surface interpolation of U
    interpolate(U)   reconIDWA;
}
```

Figure 25. Example of **interpolationSchemes** sub-dictionary

Convection Schemes

Convection Boundedness Criterion (CBC) Schemes

The following NVD (*normalized variable diagram*) based CBC schemes are particularly useful in capturing interfaces between two fluids, e.g. air and water, in the two-phase flow solver of NavyFOAM. More details of CBC schemes can be found in the Technical Description.

Compressive Interface Capturing Scheme for Arbitrary Meshes (CICSAM) The following example in Figure 26 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example the convection term is integrated using the Gauss theorem and the **CICSAM** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss CICSAM;
}
```

Figure 26. Example of **divSchemes** sub-dictionary

High Resolution Interface Capturing Scheme (HRIC) The following example in Figure 27 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example, the convection term is integrated using the Gauss theorem and the **HRIC** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss HRIC;
}
```

Figure 27. Example of **divSchemes** sub-dictionary

Modified HRIC (MHRIC) The following example in Figure 28 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example, the convection term is integrated using the Gauss theorem and the **MHRIC** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss MHRIC;
}
```

Figure 28. Example of **divSchemes** sub-dictionary

Inter-Gamma Scheme (interGamma) The following example in Figure 29 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example, the convection term is integrated using the Gauss theorem and the **interGamma** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss interGamma;
}
```

Figure 29. Example of **divSchemes** sub-dictionary

Modified inter-Gamma Scheme (interGammaM) The following example in Figure 30 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example, the convection term is integrated using the Gauss theorem and the **interGammaM** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss interGammaM;
}
```

Figure 30. Example of **divSchemes** sub-dictionary

Modified Inter-Gamma Scheme (interGammaMD) The following example in Figure 31 shows the sub-dictionary entry **divSchemes** of the system finite volume dictionary file fvSchemes that specifies the divergence schemes for the convection term in the transport equation of volume fraction (gamma). In this example, the convection term is integrated using the Gauss theorem and the **interGammaMD** scheme is used to calculate the face-center value of **gamma**. Users may refer to *OpenFOAM User Guide Ver. 1.5* Section 4.4.5 on page U-111 for more details about this sub-dictionary.

Dictionary file:

```
$(CASE_DIR)/system/fvSchemes
```

Sub-dictionary:

```
divSchemes
{
    //- convection term in gamma equation
    div(phi,gamma)          Gauss interGammaMD;
}
```

Figure 31. Example of **divSchemes** sub-dictionary

Example Results

Results are demonstrated for a fully submerged axisymmetrie body, Body-1, the KVLCC2 tanker without a free surfaee using the double body approximation, DTMB Model 5415 with both fixed sinkage and trim as well as dynamie sinkage and trim and the Joint High Speed Sealift (JHSS) eoneept surface ship with and without waterjet propulsion.

Body-1

This seetion involves using NavyFOAM's steady, ineompressible Reynolds Averaged Navier-Stokes (RANS) solver for a 3-D body-of-revolution referred to as Body-1. The RANS equations are solved using NavyFOAM's k-omega SST turbulence model. Only half the body is solved, as symmetry is assumed, and the domain is non-dimensionalized by length. The Reynolds number (*Re*) based upon the body length is 6.6 million. The boundary eonditions used for these eomputations are:

- Defined fixed turbulent quantities (k, omega, nuTilda) and veloeity (U) at inlet
- Defined pressure (p) at outlet
- Zero gradient for all quantities at farfield boundaries
- nuTilda = 0, k ~ 0, and omega set to zero gradient at the walls

A side view of the ONR Body 1 geometry can be seen below in Figure 32.

Figure 32. Body-1 geometry

Computations were done on unstruetured meshes that eontain tetrahedral (in the flow field) and prism (in the boundary layer) elements. Grids for these eomputations typieally eontained approximately 2 million eells total. The left of Figure 33 shows an unstruetured surface mesh on the body and the symmetry plane, and the right side of Figure 33 shows a surface mesh with a span-wise cross sectional cut of the volume mesh at the midbody.

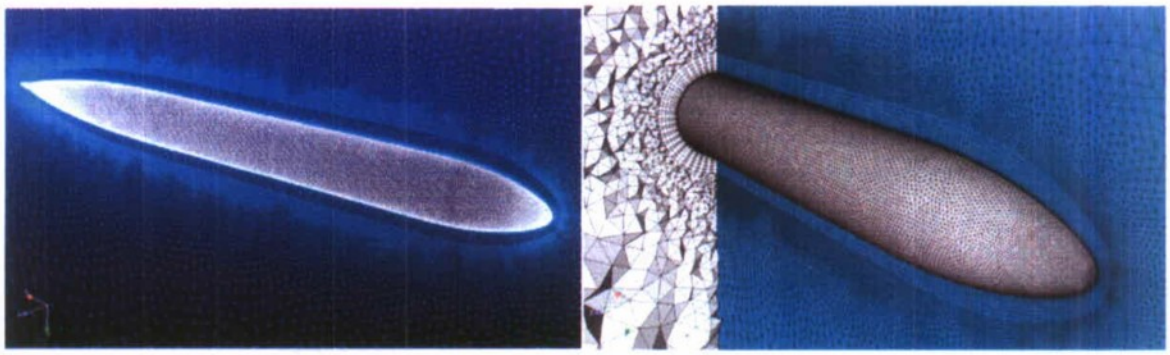


Figure 33. Surface mesh on the body and symmetry plane (left) and surface mesh with volume mesh cross cut (right)

The computational domain is split into many domains to allow the computations to be run in parallel. The domain is split using the METIS domain decomposition method. Typical steady state run times for this geometry are 3-5 hours depending on the number of domain partitions. Steady state convergence is assumed when the forces (pressure and viscous) on the body change by a negligible amount from one iteration to the next. Figure 34 shows results from NavyFOAM computations.

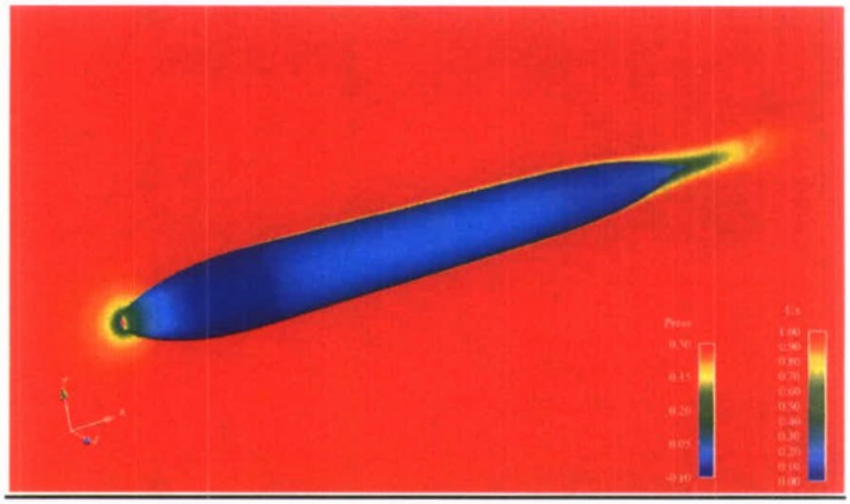


Figure 34. Axial velocity (*Ux*) contours on the symmetry plane and pressure (*Press*) contours on the hull

One can notice that the stagnation point is qualitatively predicted well at the nose of the body. The velocity slows to zero and the pressure on the body is a maximum at the nose. While the drop in pressure associated with the acceleration of the fluid around the shoulder of the bow is also predicted correctly. Computations also show the flow remaining attached along the body leaving a wake after the stern.

Figure 35 shows some of the quantitative results from the NavyFOAM computations compared to experimental measurements.

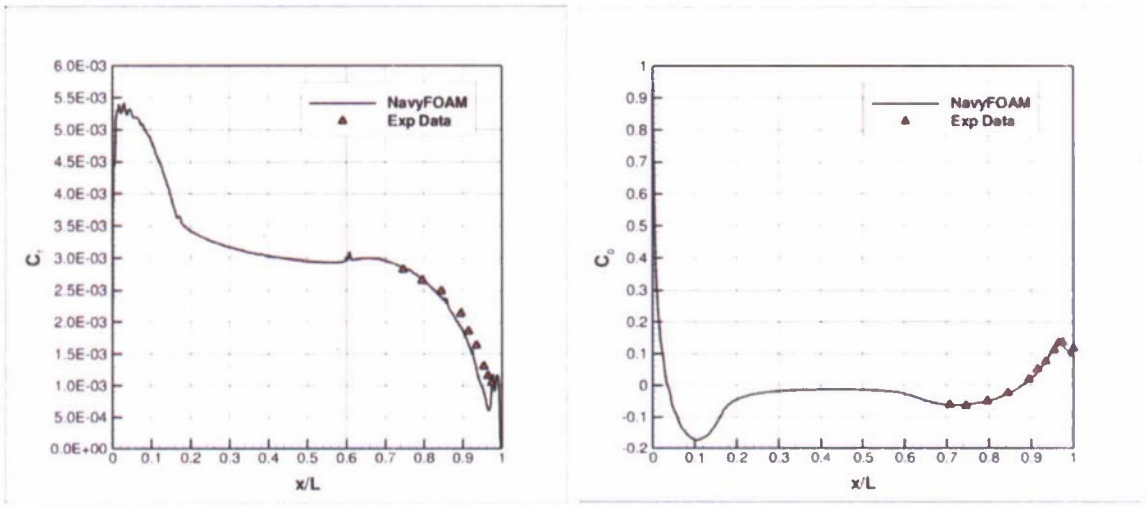


Figure 35. Skin friction coefficient (C_f) and Pressure coefficient (C_p) plotted along the length of the body

In Figure 35 there is a slight disagreement in predicted and measured C_f at the tail of the body, but the trends are matched well. The NavyFOAM C_p predictions match experimental measurements well along the body.

Figure 36 below shows comparisons of computed and measured boundary layer profiles at various locations along the body.

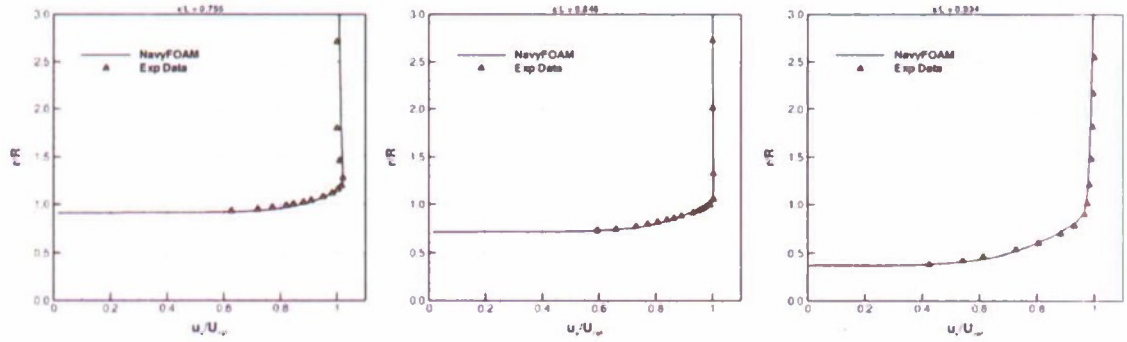


Figure 36. Axial velocity boundary layer plots at x/L = 0.755 (left), x/L = 0.846 (middle), x/L = 0.934 (right)

The axial velocities are non-dimensionalized by the free stream velocity and boundary layer lengths are non-dimensionalized by the radius. NavyFOAM computed boundary layer velocity profiles mateh experimental measurements very well at various locations along the body.

In conclusion, this effort shows that NavyFOAM has the capability to successfully predict quantitative and qualitative characteristics for a body-of-revolution, as demonstrated on the Body-1. The RANS computations were carried out successfully on multiple processors on unstructured meshes. Results for non-body-of-revolution hull forms can be found in Delaney et al.[19].

KVLCC2

The KVLCC2 is a tanker used for eode validation by many organizations and as a part of several international workshops. The stern of the KVLCC2 with predieted streamlines is shown in Figure 37, flow is from left to right.

Figure 37. Stern flow of the KVLCC2

We ran sRansFOAM for this double-body flow case in which the free surface is replaced by a symmetry plane. We deliberately ehose to use two unstructured meshes to evaluate spatial aceuraey of the predietions. One of them, shown at the left of Figure 38, is a hybrid unstruetured mesh generated using SolidMesh developed at the SimCenter of the Mississippi State University. The mesh eonsists of prisms with triangular bases near the hull surfaee and tetrahedral in the rest of the domain. The total eell eount is approximately 8 million. The other mesh, shown at the right of Figure 38, is a hexahedron-dominant unstruetured mesh generated using the snappyHexMesh utility available in the standard OpenFOAM paekage. With the latter, we put in three-levels of embedded fine-mesh bloeks to better resolve the near-body region. Its total number of elements is a little over 3 million. The near-wall mesh resolutions for both meshes are sueh that the distanee from the wall is larger than 40 wall units ($y^+ > 40$) over a large portion of the hull surfaee. So, the wall funetion approaeh was employed to provide the boundary eonditions for the momentum equations and the turbulenee equations. Wileox's k-ω model (Wileox[3]) was used for turbulenee elosure for its good traek reeord for this elass of flows (Kim and Rhee[20]).

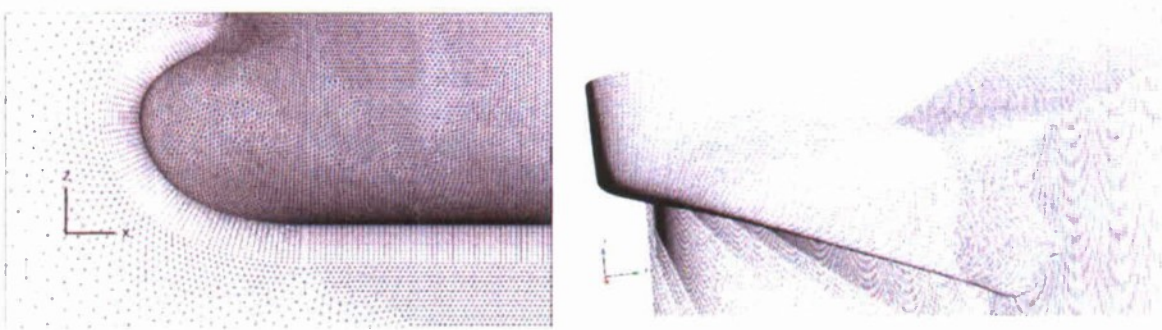


Figure 38. Grids used for the KVLCC2

The eontours of mean axial veloeity (U) at the propeller plane (x/L = 0.9823) are shown in Figure 39 for the two meshes. As ean be seen, the eharaeteristie shape of the U-eontours

("hook"-shaped or "rabbit's" ear-like) is closely captured by both meshes. The contours of turbulent kinctic energy at the samc plane are depicted in Figure 40. The region of high turbulent kinetic energy, which originates from the upstream boundary layer and overlaps with the region of low axial velocity depicted in Figure 39 is reproduced reasonably well by the computations, although their peak values are somewhat under predicted. The overall prediction accuracy shown here with the unstructured meshes used in this study is remarkably good. More details on these calculations can be found in Kim et al.[21].

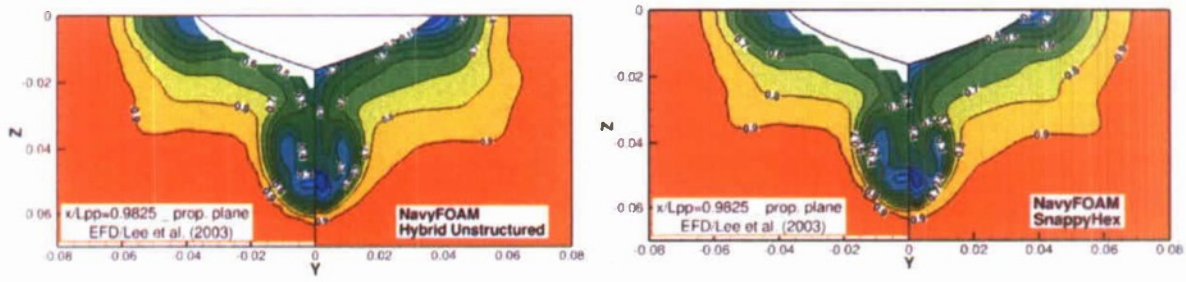


Figure 39. Contour of axial velocity at $x/L = 0.9825$ predicted on the two meshes. Top – hybrid (prism + tet) unstructured mesh; Bottom – snappyHexMesh

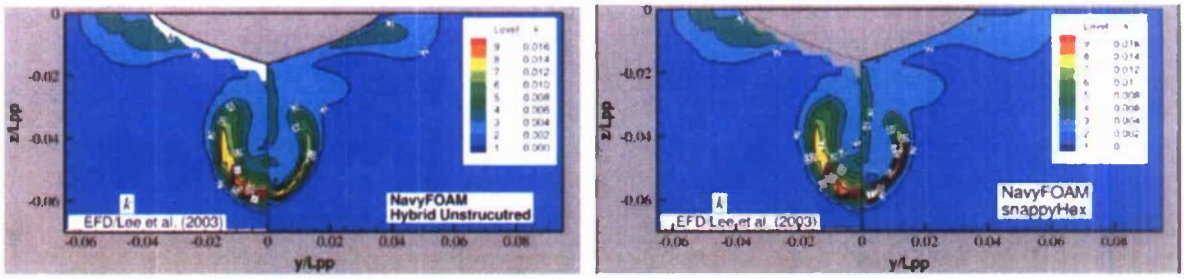


Figure 40. Contour of turbulent kinetic energy at $x/L = 0.9825$ predicted on the two meshes. Top – hybrid unstructured (prism + tet); Bottom - snappyHexMesh

DTMB Model 5415

Fixed Sinkage and Trim

These computations were carried out using ransFSFOAM for a single Froude number of 0.28 using three different isotropic eddy-viscosity based turbulence models including the realizable k-ε (Shih et al.[22]), SST k-ω (Menter[23]) and Wilcox's k-ω (Wilcox[3]) models. The computational meshes were generated using GridPro (www.gridpro.com), a commercial meshing package well known for high-quality hexahedral meshes. Great care was taken to ensure that such salient features as the hull-generated waves, the boundary layer along the entire hull, and the near-wake are adequately resolved. To check grid-dependency of the solutions, three systematically refined hexahedral meshes were used with 13 million (fine), 6 million (medium), and 3 million (coarse) elements. One of the grids used is shown in Figure 41.

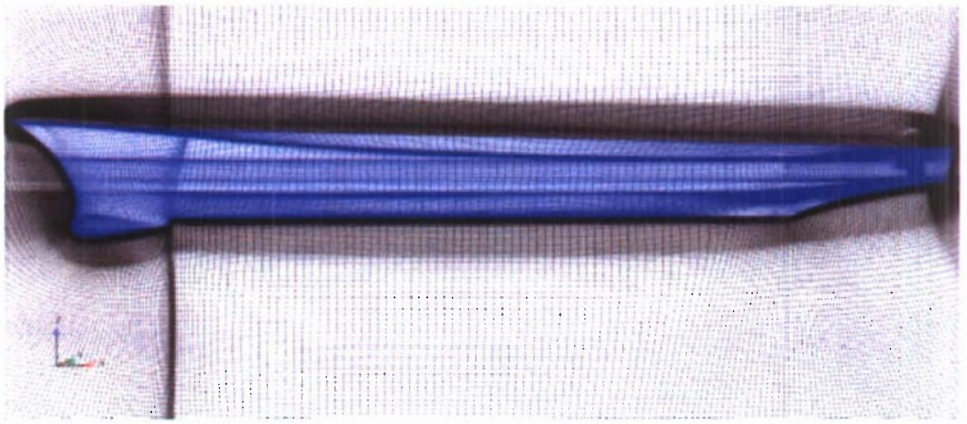


Figure 41. Grid for DTMB Model 5415

Although a time-marching, transient solution algorithm is employed in ransFSFOAM, our goal was to obtain steady-state solutions. An iterative implicit solution algorithm employed in ransFSFOAM greatly accelerated solution convergence by allowing us to use a fairly large time-step size. The transient computations were continued for sufficiently long periods of time until not only the global quantities, such as the forces and moments acting on the hull, but also other flow features like wave elevation, hull boundary layer and wake, all reach steady states.

Figure 42 illustrates the wave pattern predicted with the SST k-ω model result on the 6 million cell mesh, along with the measured one. Figure 43 shows the wave elevations along the three longitudinal cuts (y/L = 0.082, 0.172, 0.301) obtained using the SST k-ω model on all three meshes. First of all, the results indicate grid-convergence of the predictions, all of which are in excellent agreement with the data. The results obtained using three different turbulence models on the medium (6 million cell) mesh are shown in Figure 44. The differences among the three results are measurable yet small. The realizable k-ε model results appreciably deviate from the other two k-ω model results.

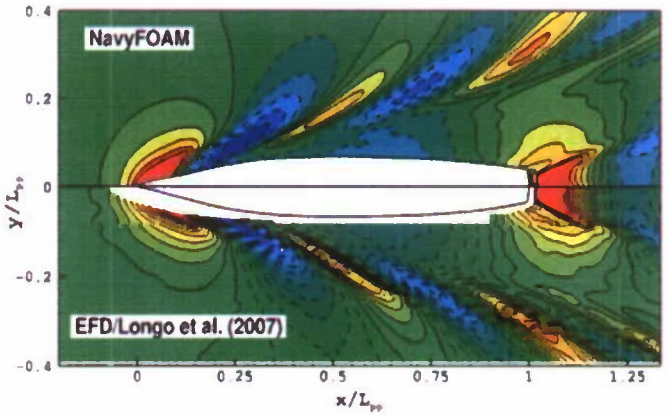


Figure 42. Contour of wave elevation for DTMB 5415 with SST k-ω model result on the 6 million cell mesh

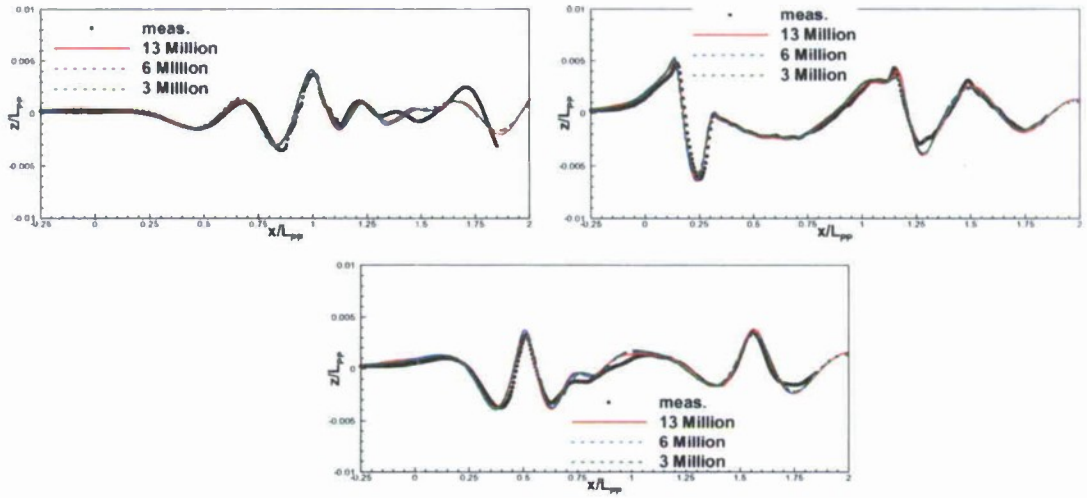


Figure 43. Wave elevations along three longitudinal cuts obtained using SST k-ω model on three different meshes: top - y/L = 0.082; middle - y/L = 0.172; bottom – y/L = 0.301

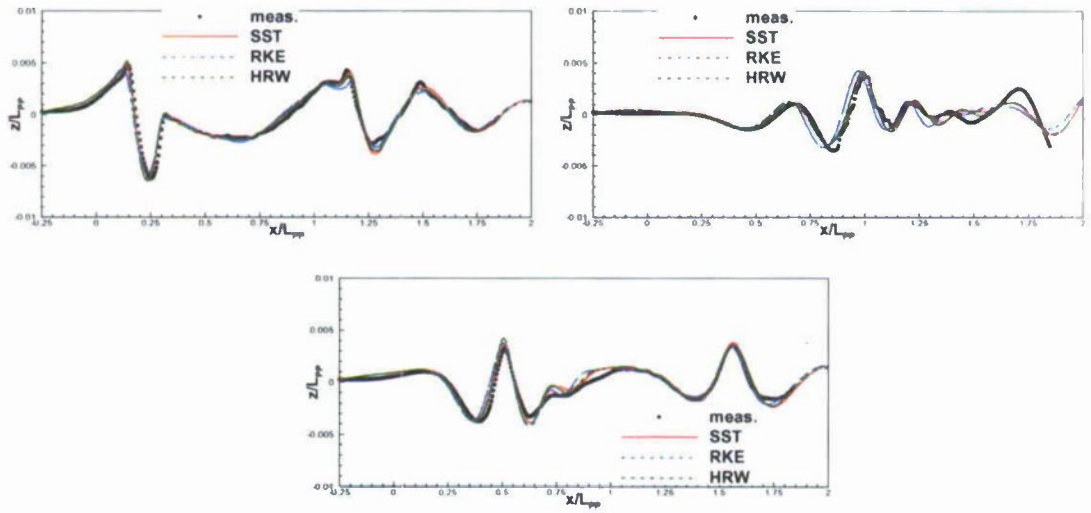


Figure 44. Wave elevations along three longitudinal cuts obtained using three different turbulence models on the 6 million cell mesh : top - y/L = 0.082; middle - y/L = 0.172; bottom – y/L = 0.301

Figure 45 shows the contours of axial velocity at x/L = 0.935 obtained using the three turbulence models on the 6 million cell mesh. All three turbulence models capture the gross feature of the boundary layer - the distorted velocity contours reflecting thickening of the stern boundary layer due to convergence of wall-limiting streamlines, and also the ensuing vortex sheet, the degree of which varies model to model. The prediction by Wilcox's k-ω model (the bottom-right figure) seems to be the closest to the measurement.

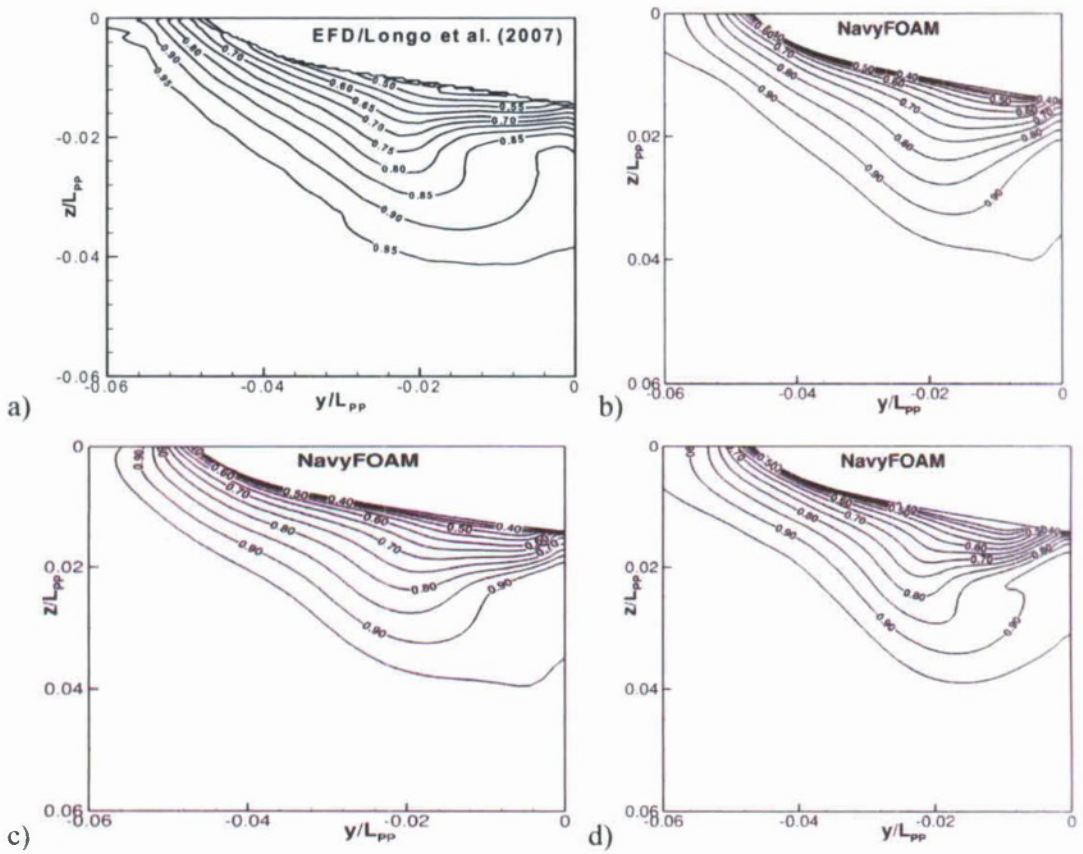


Figure 45. Contour of axial velocity (U) at $x/L = 0.935$ obtained on 6 million cells. a) measured; b) SST k-ω ; c) realizable k-ε ; d) Wilcox's k-ω

Dynamic Sinkage and Trim

The dynamic sinkage and trim computations were carried out using ransFSDyMFOAM on a 1.5 million-cell hexahedral mesh for three different Reynolds numbers, $Re = 5.96 \times 10^6$, 1.19×10^7, 1.75×10^7, and the corresponding Froude numbers, $Fn = 0.138$, 0.28, 0.41, respectively. Unsteady RANSE was solved along with the equations of 2-DOF (heave and pitch) ship motion. The transient runs were continued until the resistance, the trim angle, and the sinkage reach statistically steady states. The time-averaged resistance, trim, and sinkage are shown in Figure 46 along with the experimental data. Despite the relatively coarse mesh used in this study, the predictions are in fair agreement with the measurements. More details on these calculations can be found in Kim et al.[21].

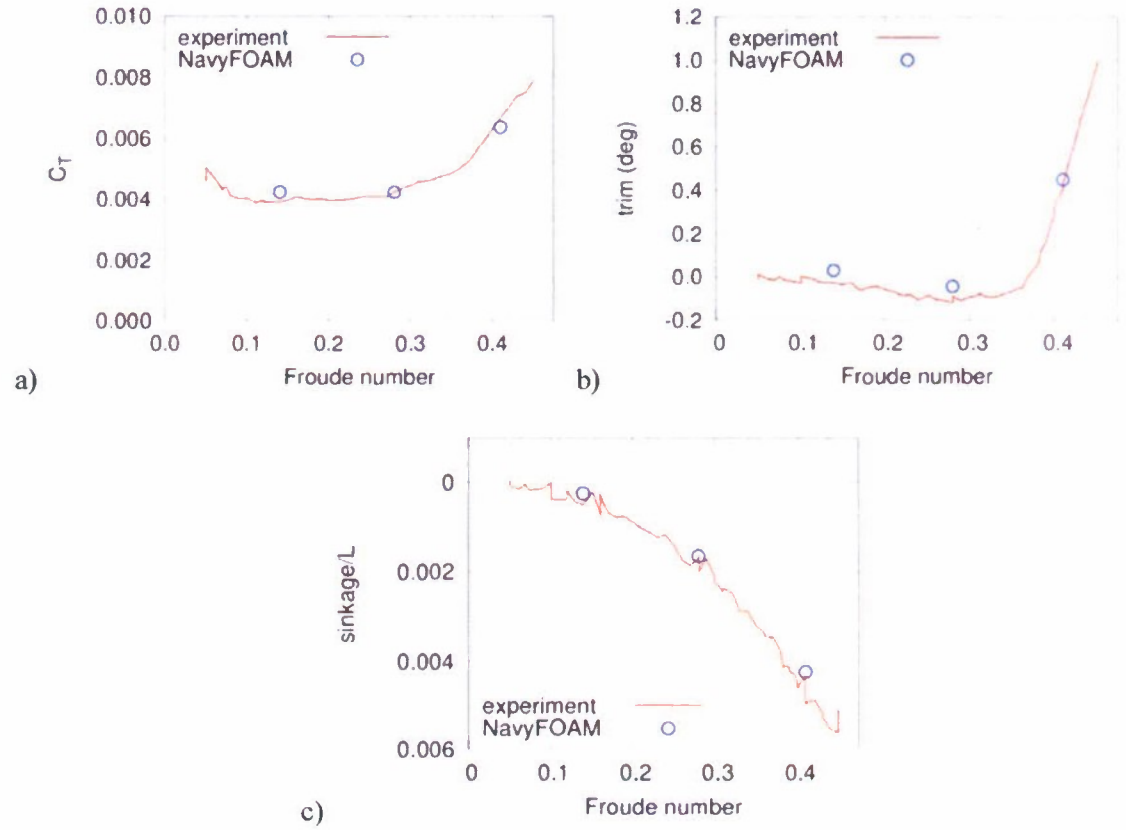


Figure 46. Prediction of a) resistance, b) trim and c) sinkage for the DTMB 5415 model

Joint High Speed Sealift (JHSS)

This section involves using NavyFOAM's multiphase, incompressible Reynolds Averaged Navier-Stokes (RANS) solver for the Joint High Speed Sealift (JHSS) concept surface ship. The JHSS concept vessel is a challenging computational case because of its complex geometry and free surface interactions with both the bow and waterjets. The RANS equations are solved using NavyFOAM's k-omega SST turbulence model. Only half the body is solved, as symmetry is assumed, and the domain is non-dimensionalized by ship length. The concept vessel is scaled from full scale by constant Froude number to a model scale. The Froude number (*Fr*) ranges from ~0.23-0.40. The air and water phases are accounted for using NavyFOAM's implicit Volume-of-Fluid (VOF) capability. The boundary conditions used for these computations are:

- Defined fixed turbulent quantities (k, omega, nuTilda) and velocity (U), and calm water volume fraction conditions (gamma) at inlet
- Defined pressure (p) at outlet
- Zero gradient for all quantities, and calm water volume fraction conditions (gamma) at far field boundaries
- nuTilda = 0, k ~ 0, and omega set to zero gradient at the walls

Side and stern views of the JHSS geometry can be seen below in Figure 47. The left side of Figure 47 shows the profile of the concept vessel including the gooseneck bow, which has

very little clearance from the free surface. The right side of the figure shows the waterjets from the stern. Both the gooseneck bow and the waterjet inlets/nozzles are complex geometry features that require special care/treatment in the meshing process.

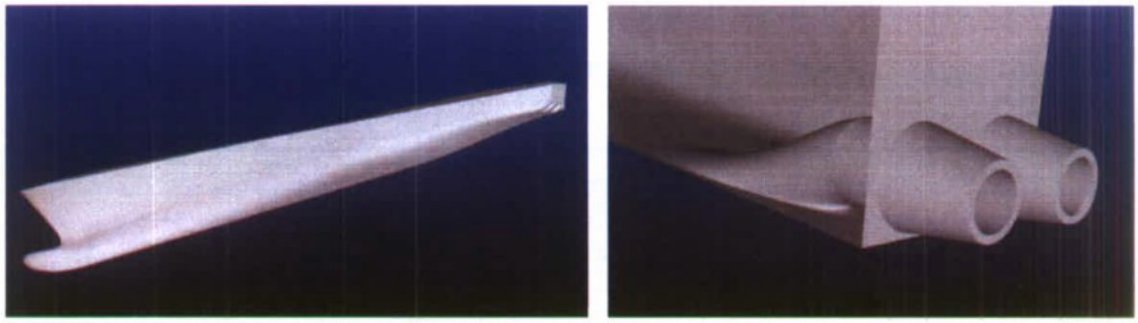


Figure 47. JHSS concept vessel geometry

Unpowered Bare Hull Computations

Initially, computations are done with the bare hull (no waterjets included in the model) to test NavyFOAM's multiphase sinkage and trim capability. Later in the report powered computations involving the waterjets will be discussed. Figure 48 shows the structured surface mesh on the bare hull used for these calculations. The surface mesh on the JHSS displays mesh refinement around the free surface to capture free surface disturbances. The meshes used for this study were typically on the order of 2-4M cells total.

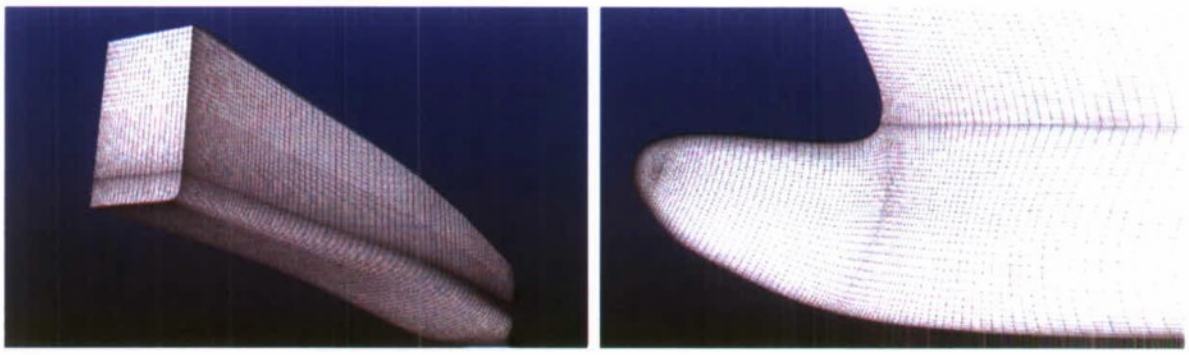


Figure 48. JHSS structured surface mesh on the bare hull

The computational domain is split into many domains to allow the computations to be run in parallel. The domain is split using the METIS domain decomposition method. Typical steady state run times for this geometry are 48-72 hours depending on the number of domain partitions. Steady state convergence is assumed when the forces (pressure and viscous) on the body and sinkage and trim values change by a negligible amount from one iteration to the next. Figure 49 shows wave profile results from NavyFOAM computations.

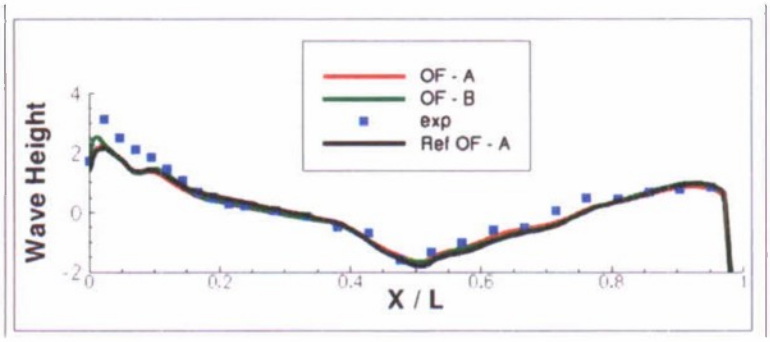


Figure 49. JHSS wave profile on the hull for various NavyFOAM meshes and experimental measurements, scaled up to full scale (full scale LBP ~950 ft)

Figure 49 shows results from a NavyFOAM grid study, and experimental measurements for a slightly different JHSS model. A grid resolution study and a blocking study resulted in 3 different meshes (OF-A, OF-B, and Ref OF-A). The grid study showed that NavyFOAM gave relatively consistent results for all three meshes, thus the grid scheme shown in Figure 48 is used for all results in this report. Figure 49 also shows that the NavyFOAM predicted wave profiles match experimental measurements well. However, one can see that the bow wave is slightly under predicted. This is most likely due to a lack of grid resolution in the bow area and/or need for a sharper volume fraction discretization scheme. Nevertheless the differences are relatively small considering the ship length is 950 feet and the bow wave is under predicted by ~1 foot.

Figure 50 shows some axial velocity boundary layer profiles upstream of where the waterjet inlet would be located (inboard on the left and outboard on the right). NavyFOAM results (*OF*) are compared to both experimental measurements (*Exp*) and previous double body calculations with another RANS solver (*TENASI*). The NavyFOAM boundary layers match experimental measurements very well. These boundary layer plots are important because powering predictions ultimately depend on (amongst other things) accurate prediction of the flowfield upstream of the waterjet inlets.

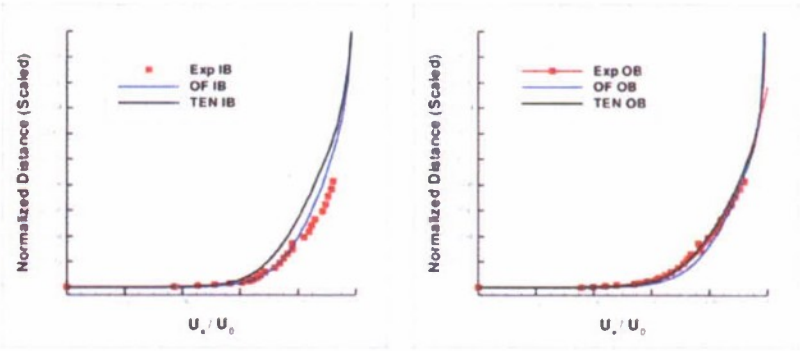


Figure 50. Inboard (left) and outboard (right) axial velocity boundary layer plots for NavyFOAM free surface computations (OF), TENASI double-body computations (TEN), and experimental measurements (Exp)

Figure 51 shows sinkage and trim values over the course of a NavyFOAM run for three Froude number cases. Each plot shows a consistent pattern for both sinkage and trim for all run times, thus showing that the multiphase solver is relatively robust, and wild swings in sinkage and trim over the course of a run are not predicted.

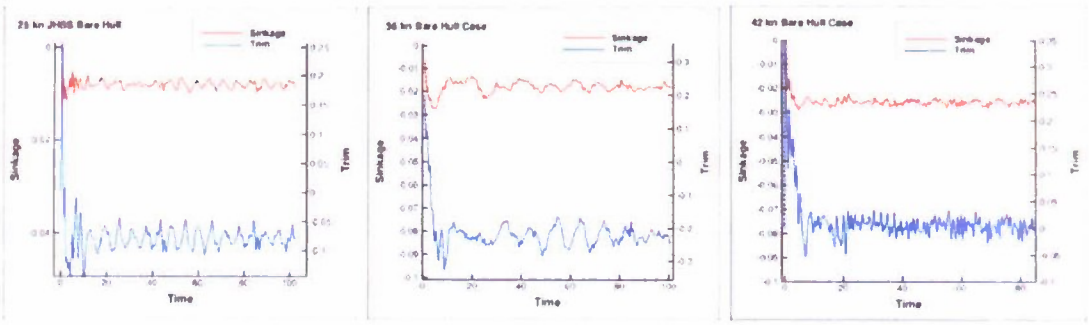


Figure 51. Sinkage and trim run time values plotted for three different Froude numbers

Figure 52 shows free surface plots for the bare hull configuration under fixed (to the design point) and free sinkage and trim cases. These plots show that the predicted wave profiles for the free to sink and trim case are similar to the profiles predicted under fixed conditions.

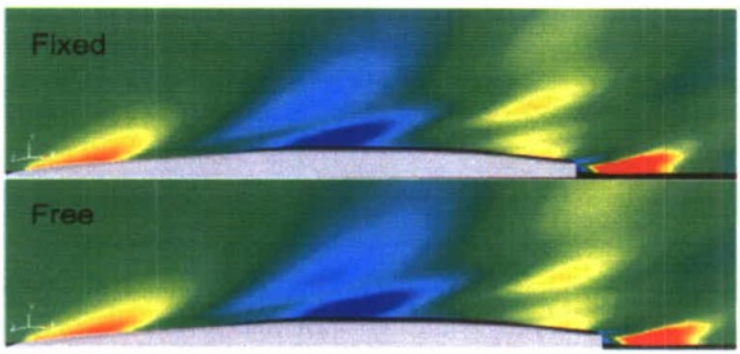


Figure 52. Fixed and free sinkage and trim NavyFOAM free surface plots colored by wave elevation

Figure 53 shows total resistance predictions on the body over various Froude numbers for both NavyFOAM and experimental measurements. One can see that NavyFOAM predicts drag on the body extremely well for all Froude numbers as compared to experiment. These successful predictions are important because resistance prediction is a key to powering predictions.

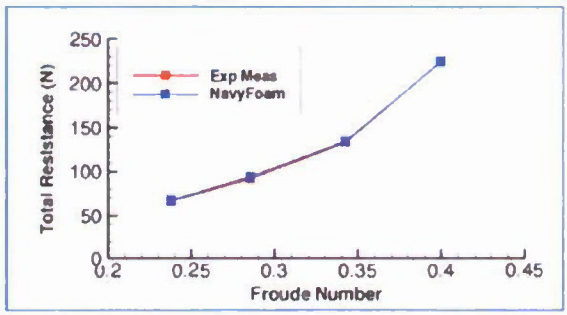


Figure 53. JHSS resistance for various Froude numbers predicted by experiment and NavyFOAM

Figure 54 shows sinkage (top) and trim (bottom) comparisons to experimental measurements for various Froude numbers. NavyFOAM sinkage predictions match experimental measurements very well, with a slight discrepancy at the highest Froude number. NavyFOAM trim predictions also match experimental measurements well over the range of Froude numbers, with a slight discrepancy at the two highest Froude numbers. Although the trim differences may look large they only differ by fractions of a degree. In both the sinkage and trim cases the overall trends are predicted correctly.

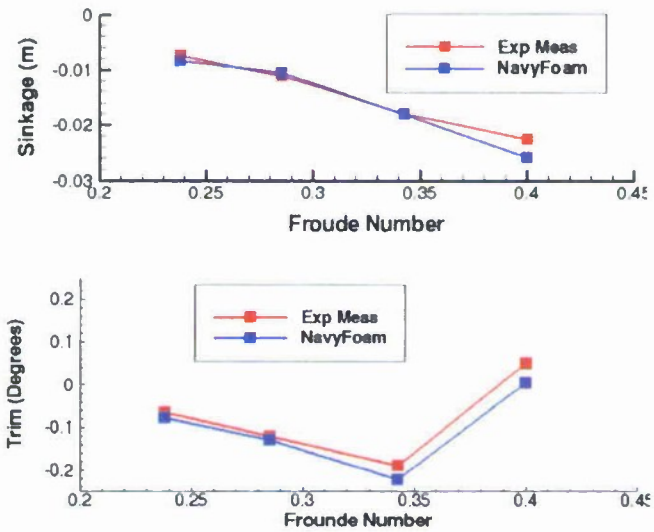


Figure 54. JHSS barc hull sinkage (top) and trim (bottom) predictions for various Froude numbers

Powered Computations with Waterjets

The rest of this section will discuss powered JHSS conditions. The powering model described below is for the JHSS concept vessel fixed at design sinkage and trim. The waterjet pumps are simulated by a body force model in NavyFOAM, which imposes a pressure jump over a volume region as specified by the user. Figure 55 shows the surface mesh on the waterjet region of the JHSS. The complexity of the waterjet inlet geometry led to the desire to use unstructured elements (tetrahedral and prism) to grid around the waterjet inlets. NavyFOAM's Generalized Grid Interface (GGI) capability allows us to combine the structured mesh used for the bare hull computations with an unstructured grid around the waterjet inlets. Figure 55 shows the different structured and unstructured surfaces on the hull.

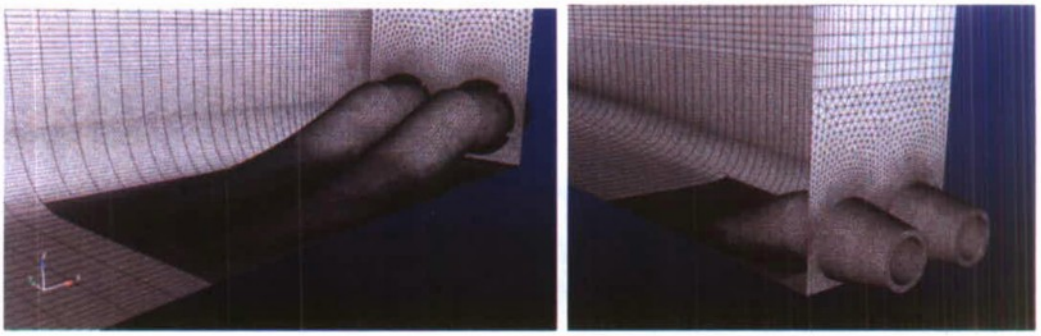


Figure 55. Surface mesh at the stern showing GGI region around waterjets

Powered computations are carried out similar to the process described above for the bare hull case (domain decomposition, steady state criterion, etc.). One additional steady state criterion is added for the powering case: the resistance on the body must match an artificial tow force (to match experimental tow tank results) and the thrust in the waterjets provided by the body force model to simulate self propulsion. Figure 56 shows axial velocity contours through an inboard waterjet inlet. The right side of Figure 56 shows the volume mesh at this cross cut. One can see that the flow remains smooth through the GGI interfaces, thus NavyFOAM's GGI capability successfully handles these complex flow interfaces.

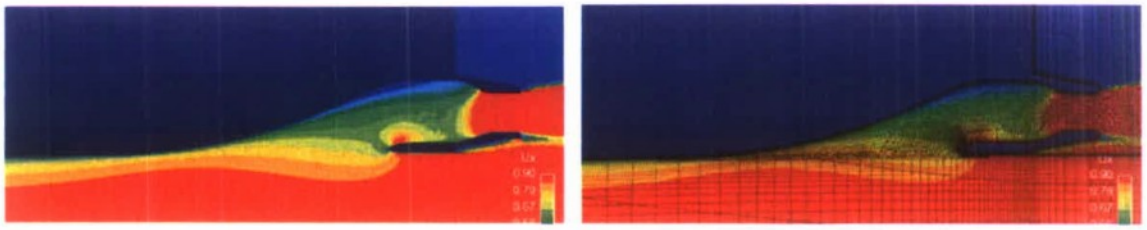


Figure 56. Axial velocity contours through the GGI modeled waterjet without (left) and with (right) volume mesh overlayed

Figure 57 shows axial velocity (Vx) contours predicted by NavyFOAM for the inboard (right) and outboard (left) waterjets just upstream of where the waterjet pumps (or body force model in NavyFOAM's case) would reside. The qualitative look of the flowfield upstream of the pump matches previously validated TENASI computations very well. It is important to accurately capture the flow upstream from the pump as it is a key factor in final power (DHP) predictions. NavyFOAM predicted self propulsion at a thrust of 163 N, while experimental tests resulted in a self propulsion point of 153 N. There is a slight discrepancy between experiment

and NavyFOAM self propulsion points, but they are still quite close considering the different methods (tow tank experiments vs. computations) used in obtaining the results.

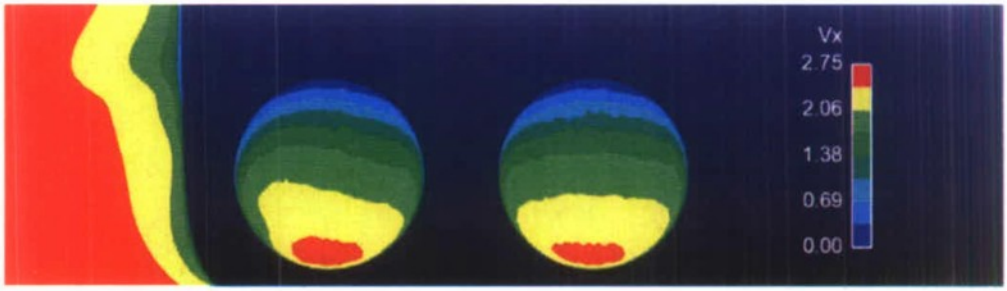


Figure 57. Axial velocity contours inside the waterjets

Figure 58 shows the free surface colored by wave height for the powered JHSS as predicted by NavyFOAM. The wave pattern is similar to that found in the bare hull case except in the stern region, as it should be. The rooster-tail at the stern is captured correctly, and the affect of the flow exiting the waterjets can be seen.

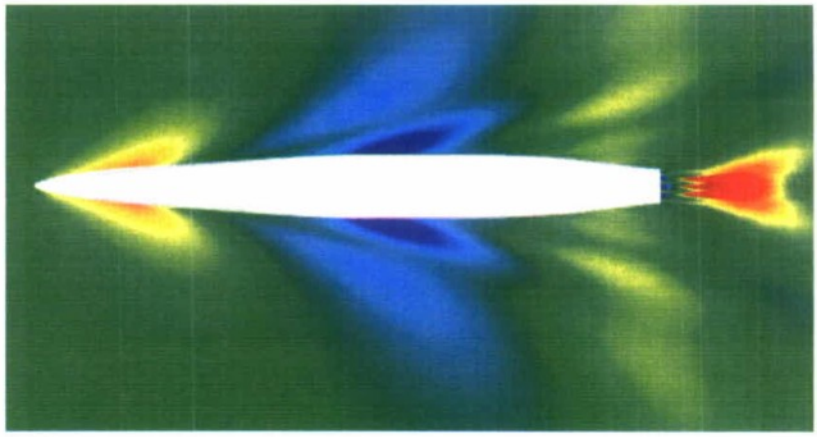


Figure 58. Powered JHSS free surface plot colored by wave elevation

Figure 59 shows a photograph of the flow exiting the waterjets during tow tank tests compared to NavyFOAM's post-processed results. One can see that the NavyFOAM computations predict the complex physics taking place at the stern very well. The interaction between the rooster-tail shooting out from underneath the stern and the flow exiting the waterjets is captured very well in NavyFOAM.

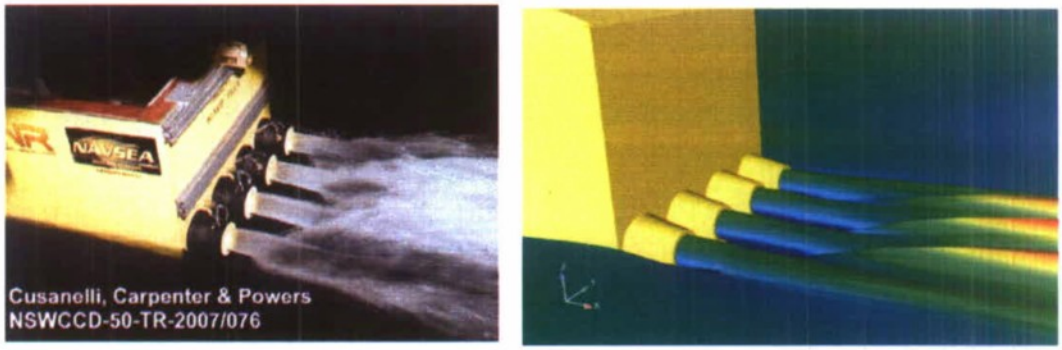


Figure 59. Experimental photograph (left) and NavyFOAM post-processed JHSS powered stern

In conclusion, NavyFOAM displays the capability to accurately predict complex physics for surface ships. The JHSS concept vessel is an especially complex surface ship due to the gooseneck bow and the waterjet inlets/powering. Initially, the hull resistance, sinkage and trim of the bare hull case are predicted with the hull free to sink and trim. Navy FOAM results match experimental measurements very well for various Froude numbers that take the concept vessel through different ship attitudes. Finally, self propulsion is predicted with a body force model (in place of the actual waterjet pumps) providing thrust that balances out the model's resistance. NavyFOAM powering predictions match experimental measurements well.

Summary

This report provides a guide to NavyFOAM V1.0, which is based on the OpenFOAM open source software. A brief technical description of the code is given with an emphasis on those changes made for NavyFOAM V1.0 that differentiates it from the standard OpenFOAM offering. More details on OpenFOAM specifically can be found in the OpenFOAM guides referenced in this report. Complementing the technical description of NavyFOAM changes there is a User's Guide section to help users of NavyFOAM properly implement the use of these improvements. Results are presented for a variety of Navy relevant configurations including a fully submerged axisymmetric body, a tanker, DTMB Model 5415 (pre-contract design for the DDG-51) and the Joint High Speed Sealift (JHSS) concept. Results for all cases compare will with experimental data. Results are also presented for a variety of grid types and turbulence models providing some indication of the capabilities available with the code. Finally, several tutorials as well as other information that is directly aimed at helping users successfully use the code are also provided.

Appendix A: Supplemental User's Guide

This section is meant to supplement the original OpenFOAM User's Guide. For more detailed information on the OpenFOAM code and settings consult the User's Guide: http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf.

Standard solvers that are used:

- icoFoam (transient, laminar, incompressible, single phase)
- simpleFoam (steady, RANS, incompressible, single phase)
- interFoam (transient, RANS, incompressible, multi-phase)

Files Contained in the system Directory include:

(1) controlDict

(2) decomposeParDict

(3) fvSchemes

(4) fvSolution

controlDict

Dictionary that controls run parameters and output data from the run.

Application: here you specify the solver used (i.e. simpleFoam, icoFoam, etc.). This is not essential. Specifying the executable here doesn't seem to have any control over the run.

StartFrom: here you specify when to start the run from. Options are:

firstTime – start from the earliest time directory available

startTime – start from the time specified by *startTime* on the next line

latestTime – start from the most recent time directory available.

startTime: enter the time to start the run from when *startFrom ? startTime;* is selected.

StopAt: here you specify when to stop the application. Options are:

endTime – stop run at time specified by *endTime* on the next line

writeNow – stop the run at the next iteration and write out the last time step

noWriteNow – stop the run at the next iteration and don't write time info at last time step

nextWrite – stop the run at the next scheduled write time specified by *writeControl*

endTime: time when run will stop. Only valid when *stopAt ? endtime;* is selected

deltaT: here you specify the run time step. This is typically *1* for steady runs and for unsteady runs this is superceded by *maxCo*, which is discussed below.

WriteControl: here you specify when/how to output information. Options are:

timeStep – writes data every *writeInterval* time steps

runTime – writes data every *writeInterval* seconds of run time

adjustableRunTime – this is the same as runTime, except it will adjust the time steps to coincide with the *writeInterval*

cpuTime – writes data every *writeInterval* seconds of CPU time

clockTime – writes data every *writeInterval* seconds of real time

writeInterval: scalar data that specifies time output from *writeControl*

purgeWrite: this is an integer value that specifies how many output time directories are stored. When the max number of *purgeWrite* directories have been written, the newest time data will over write the oldest time directory. A value of *0* means that no time data will be overwritten.

WriteFormat: here you specify the format of the output data. Options are:

ascii – ASCII data with the amount of significant figures specified by *writePrecision*

binary – Binary data

ascii is typically used

writePrecision: number of significant figures ASCII *writeFormat* is written out to.

WriteCompression: here you specify the compression (if any) of the output files. Options are:

uncompressed – No data compression

compressed – gzip compression

compressed has typically been used

timeFormat: here you specify the format for the time directory names. Options are:

fixed – all time directories are written in fixed format (i.e. 123.456)

scientific – all time directories are labeled in scientific format (i.e. 1.23456e+03)

general – specifies scientific format of exponent is less than -4, otherwise fixed

general has typically been used

timePrecision: here you specify the number of significant figures for *timeFormat* time directory labels. *6* is the default and is typically used.

GraphFormat: here you specify the graph data written by an application. Options are:

raw – data in ASCII format in columns

gnuplot – data in gnuplot format

xmgr – data in Grace/xmgr format

jplot – data in jPlot format

typicall raw is specified, but this depends on the user preference.

RunTimeModifiable: here you specify yes to have the directories (including *controlDict*) read at the beginning of each time step, or no to have the run proceed without rereading directories. This is typically set to yes, and should remain so to avoid confusion.

AdjustTimeStep: select yes for an adjustable time step (whose size is determined by *maxCo and maxDeltaT*), or no for a constant time step as specified by *deltaT*.

MaxCo: This adjusts the time step to achieve the specified maximum CFL number. This value is typically low *O(1)* at the beginning of a run, and ramped up *O(10)* once the solution becomes more stable.

MaxDeltaT: this value is a limit to the maximum time step achievable, when a *maxCo* is specified.

** At the bottom of the *controlDict* (or anywhere within) you can specify additional libraries or functions to be loaded at run-time. Turbulence model and dynamic mesh motion *libs*, and force *functions* are common examples of what have been used.

decomposeParDict

Dictionary that contains all input information on domain decomposition for parallel processing runs.

Subdictionary: numberOfDomains

Here you specify the number of partitions you want your grid to get split up into.

Subdictionary: method

Here you specify the method of domain decomposition. Options are:

simple – domain is decomposed into volumes that are similar in all coordinate directions.

hierarchical – user can specify the order of direction for simple decomposition.

scotch – attempts to minimize the number of geometric boundaries.

metis – similar to scotch, but is not free for commercial use, it will eventually be discontinued.

manual – user manually specifies decomposition of each cell to a processor.

metis has traditionally been used during testing, occasionally *simple* decomposition is used. User should use *metis* except for special circumstances.

Subdictionary: simpleCoeffs

Here you specify *n* number of processors in each direction to decompose domain, and cell skew factor (*delta*) for decomposition. This is only necessary for *simple* domain decomposition.

Subdictionary: hierarchicalCoeffs

Here you specify *n* number of processors in each direction to decompose domain, and cell skew factor (*delta*), and the *order* of directions (i.e. xyz or zyx) for decomposition. This is only necessary for hierarchical domain decomposition.

Subditionary: scotchCoeffs

Here you specify the weighting factors (*processorWeights*) for each individual processor. The numbers for each processor are normalized, so any values can be accepted, no matter the

sum. You can also specify the *strategy*, but it is not clear what this value means. This is only necessary for scotch domain decomposition.

Subdictionary: metisCoeffs

Here you specify *processorWeights*, as described above in *scotchCoeffs*. This is only necessary if you specify metis decomposition.

Subdictionary: manualCoeffs

Here you specify the name of a data file that contains processor allocations for each cell. This is only necessary if you specify manual decomposition.

Subdictionary: distributed

This is an optional subdictionary, where you state yes or no, whether there is geometry data in any other directories that needs to be decomposed with the current directory.

Subdictionary: roots

If you chose *yes* to *distributed*, here you list the address(es) to additional directories.

fvSchemes

Dictionary that contains numerical scheme input: interpolation methods, temporal and spatial discretization information

Subditctionary: ddtSchemes

This subdictionary contains first time derivative discretization method

Euler (1st order implicit) is the only method that has been consistently used

Subdictionary: gradSchemes

This subdictionary contains discretization information for the gradient terms

Gauss linear and *leastSquares*. The Gauss linear method has been used most frequently, yielding consistent results. The *leastSquares* method is believed to be more accurate when calculating the gradient on non-uniform meshes, but bugs were encountered early in the V & V process. Improvement of the *leastSquares* gradient method has been an ongoing effort.

Limiting is available (i.e. *Gauss linear limited*); however, this was proven to negatively affect the solution during the V & V process.

Subdictionary: divSchemes

This subdictionary contains discretization information on the divergence terms.

Div(phi,U) is commonly refered to as the convection term in the momentum equation. Typically, a 2nd order upwind is used for this term,*Gauss linearUpwind cellLimited Gauss linear 1.0.*

Turbulent quantities (nuTilda, k, omega, epsilon, etc.) are usually solved 1st order upwind (*Gauss upwind*), but it may be necessary to solve the 2nd order upwind (*Gauss linearUpwind cellLimited Gauss linear 1.0*).

Multiphase gamma terms…

*Note: There will be an error message if the term *div((nuEff*dev(grad(U).T())))* is left out, or if a divScheme is applied to this term. Instead this term needs to be included with a gradient scheme diseretization. *Gauss linear* is the most common scheme used with this term.

Subdictionary: laplacianSchemes

This subdictionary contains discretization information for the laplaeian terms.

Gauss linear corrected has been the standard seheme used. Limiting ean be used (*Gauss linear limited 0.0* = uncorrectcd and *Gauss linear limited 1.0* = eorreeted). The eorreetion refers to treatment of non-orthogonal terms. An uneorreeted solution (*Gauss linear limited X.X* with *X.X* < 1 will not eonverge to the same, more aeeurate, solution as *Gauss linear corrected*).

Subdictionary: interpolationSchemes

This subdietionary eontains information on interpolation that is usually from cell center to cell face.

Linear has been the standard scheme used; however, reconCentral seheme is being developed and is believed to be more accurate for meshes with signifieant non-orthogonality (unstructured meshes).

Subdietionary: snGradSehemes

This subdietionary contains information on surfaee normal gradient terms. This term speeifies the portion of the gradient at a cell's face that is normal to the faee.

corrected has been the standard seheme used for this subdictionary and other schemes have not been used signifieantly.

Subdietionary: fluxRequired

This subdictionary contains information for variables whose flux is calculated in the applieation.

The flux is required for pressure (*p* and *pd*) for most ealculations, beeause a pressure equation is solved. Thus the default is usually set to *no* and the variable *p* or *pd* is speeified in the subdietionary, so that the flux is ealeulated for the pressure.

fvSolution

Dietionary that eontains algorithm and linear system solvers information, sueh as solver settings and toleranees for convergenee. The segregated solver variables and solution algorithms will vary with ehoiee of problem solver executable (i.e. simpleFoam and interFoam require different settings).

solvers contains the linear system of equation algorithms and tolerances for all the variables (OpenFoam uses a segregated solver).

The common linear solvers tested and used are conjugate gradient solvers (PCG/PBiCG) and multi-grid solvers (GAMG/AAMG).

Many tolerance settings have been used, but eommon values tested have been:

P – tolerance 1e-10; relTol 0.01; and minIter = 1;

U, gamma, and turbulent quantities - toleranee 1e-07; relTol 0.0; and minIter = 1;

The tolerance criterion is satisfied when the residual for the linear solver reaches the prescribed value. The relTol criterion is satisfied when the linear solver residual has dropped by the order of magnitude specified by: 1.0/relTol (i.e. relTol = 0.01 corresponds to the residual dropping two orders of magnitude or 10^2).

Note: be sure to use the *minIter=1;* command for all the linear solvers, as some solution tolerances may be set such that variables will stop iterating prematurely in the solution process, thus leading to inaccurate solutions that may look OK.

Pressure algorithms (**SIMPLE** and **PISO**) -

For simpleFoam (steady state, single phase, RANS solver) the solver variables needed are p, U, turbulent quantities (nuTilda for SA, k and omega for SST or WKO, k and epsilon for k-epsilon, etc.).

The pressure algorithm should be **SIMPLE**, which includes:

nNonOrthogonalCorrectors #; Where the # selected will determine the number of additional pressure solver loops. For example, for a value of # = 1, the solver will iterate over the pressure equation twice. For runs with meshes of good quality these additional loops are not needed. However, for meshes containing a lot of skew or nonOrthogonality, values between 1 and 3 will add stability (as well as additional computational time) to the solution.

nCorrectors?

pRefCell and *pRefValue* must both be set or errors will occur. These values represent what the reference pressure value (most likely free-stream) and a cell number where this value occurs (a cell located in the free stream). All previous runs have used values of *0* for both without problem.

For unsteady and pseudo time-marching steady solvers like interFoam and ransNavyInterFoam

The pressure algorithm should be **PISO**, which includes:

nNonOrthogonalCorrectors #; ...

relaxationFactors contains the under-relaxation values for all the linear solver variables (U,p,pd,omega,k,gamma, etc.). The relaxation factors correspond to :

0.0= Fully relaxed

0.0 < # < 1.0 = variable under-relaxation

1.0 = No relaxation

For all implicit and most explicit runs, under-relaxation should be used on all variables to varying degrees. The pressure terms (p for simpleFoam, pd for other solvers) require more under-relaxation than other variables and this value is usually very important to solution stability. It is not uncommon to start a calculation out with a value of 0.1 or 0.2 and wait for the initial solution to develop and then ramp the value up progressively to 0.3. The velocity (U) usually starts around 0.5 and ramps up to 0.7 or 0.8 as the solution stabilizes. The turbulent quantities and gamma usually affect the solution startup less and typical values are 0.6 and 0.7-0.8 respectively.

Appendix B: Utility Programs

Several programs have been written to simplify post-processing NavyFOAM output data. Descriptions of their use are given here.

dataFunkyFieldComparison

Description

This utility reads the scalar and vector volume field from FOAM numerical solution data files and compares them with the analytic solutions specified by a dictionary file named *funkyFieldsDict* stored under directory *$CASE_DIR/system*. The utility will first match the field names specified in *funkyFieldsDict* with those stored in the solution files and then calculate the L_∞ and L_2 norm of the error and send the report to standard output (stdout). The error will only be calculated if the field name is found in both *funkyFieldsDict* and solution files.

This utility is part of NavyFOAM.

Usage

The command line usage looks like

```
dataFunkyFieldComparison [-case dir] [-time time] [-latestTime]
```

The optional options are

-case dir: specifies the case directory;

-time time: selects the time step;

-latestTime: selects the latest time step.

Without any option, the utility will read the data files for all time steps stored under the current case directory.

Installation

1) Create a working copy using svn checkout. The recommended local directory to checkout the package is

NavyFOAM/applications/utilities/postProcessing

```
svn checkout svn+ssh://blackwater/5700/NavyFOAM/IntegrationBranches/NavyFOAM-1.5-dev-rev995/NavyFOAM/applications/utilities/postProcessing/dataComparison
```

2) Go to directory *./dataComparison/dataFunkyFieldComparison* and compile the package:

```
wmake
```

The compiler may generate some warning message which can be ignored in this case. The generated executable file *dataFunkyFieldComparison* can be found in a user application binary file directory specified by *$FOAM_USER_APPBIN*.

Warning: To compile this utility at least version 2.1 of *Bison* has to be installed. Check with

```
bison –V
```

on the command line before trying to compile it. Go to http://www.gnu.org/software/bison/ for more information regarding *Bison*.

Expression Syntax

The example below shows some useful expression syntax. The most complete documentation of the expression syntax is the source file for the Bison-grammar in the package:

ValueExpressionParser.yy
ValueExpressionLexer.ll

Example

The dictionary file *funkyFieldsDict* used in a Taylor vortex test case is shown below:

```
FoamFile
{
  version    2.0;
  format     ascii;
  class      dictionary;
  object     funkyFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

expressions
(
  TaylorVortexVelocity
  {
    field U;
    expression "vector(sin(pos().x)*cos(pos().y), -cos(pos().x)*sin(pos().y), 0)*exp(-0.2*time())";
  }
  TaylorVortexPressure
  {
    field p;
    expression "0.25*(cos(2.*pos().x)+cos(2.*pos().y))*pow(exp(-0.2*time()),2)";
  }
);
```

The above dictionary file specifies the analytic solution of the Taylor-Green vortex problem:

$$p(x,y,t) = \frac{1}{4}(\cos 2x + \cos 2y)F^2(t)$$

$$u(x,y,t) = \sin x \cos y\, F(t)$$

$$v(x,y,t) = -\cos x \sin y\, F(t)$$

with $F(t) = e^{-2\nu t}$ and $\nu = 0.1$.

Suppose we already have the solution files at time = 1 as the latest time step stored under $CASE_DIR/1. The command line

```
dataFunkyFieldComparison –latestTime
```

will yield the output shown below:

```
...
##################################################
  Time = 1
##################################################
...
==========================================================
 volScalarField: p
 ----------------------------------------------------
  max error occurs at cell: 0
  maxErr = 0.000409110343
  rmsErr = 0.000137916310931
==========================================================
...
==========================================================
 volVectorField: U
 ----------------------------------------------------
  max error in magnitude occurs at cell: 12216
  maxErr = 6.79861100733e-05
  rmsErr = 2.44996079401e-05
 ----------------------------------------------------
  max error in X-component occurs at cell: 7680
  maxErr = 3.4175343e-05
  rmsErr = 1.44738699513e-05
 ----------------------------------------------------
  max error in Y-component occurs at cell: 12217
  maxErr = 6.7017086e-05
  rmsErr = 2.1343033345e-05
 ----------------------------------------------------
  max error in Z-component occurs at cell: 5911
  maxErr = 0
  rmsErr = 0
==========================================================
```

For a scalar volume field such as the pressure field, the maximum error (maxErr) and root-mean-squares error (rmsErr) are calculated as follows:

$$\text{maxErr} = L_\infty(p^n - p_0^n) = \max_{1 \le i \le N_P}(|\, p_i^n - p_{0,i}^n \,|)$$

$$\text{rmsErr} = L_2(p^n - p_0^n) = \sqrt{\frac{1}{N_P}\sum_{i=1}^{N_P}(p_i^n - p_{0,i}^n)^2}$$

where the subscript "0" denotes the analytic solution, the superscript "n" represents the n-th time step, and N_P is the total number of cells for cell-centered finite volume method.

For a vector volume field such as the velocity field, the maxErr and rmsErr are calculated for the vector magnitude:

$$\text{maxErr} = L_\infty(|\vec{V}|^n - |\vec{V}_0|^n) = \max_{1 \le i \le N_P}\left(\left|\,|\vec{V}_i|^n - |\vec{V}_{0,i}|^n\,\right|\right)$$

$$\text{rmsErr} = L_2(|\vec{V}|^n - |\vec{V}_0|^n) = \sqrt{\frac{1}{N_P}\sum_{i=1}^{N_P}\left(|\vec{V}_i|^n - |\vec{V}_{0,i}|^n\right)^2}$$

and for each component:

x-component

$$\text{maxErr} = L_\infty(u^n - u_0^n) = \max_{1 \le i \le N_P}(|u_i^n - u_{0,i}^n|)$$

$$\text{rmsErr} = L_2(u^n - u_0^n) = \sqrt{\frac{1}{N_P}\sum_{i=1}^{N_P}(u_i^n - u_{0,i}^n)^2}$$

y-component

$$\text{maxErr} = L_\infty(v^n - v_0^n) = \max_{1 \le i \le N_P}(|v_i^n - v_{0,i}^n|)$$

$$\text{rmsErr} = L_2(v^n - v_0^n) = \sqrt{\frac{1}{N_P}\sum_{i=1}^{N_P}(v_i^n - v_{0,i}^n)^2}$$

z-component

$$\text{maxErr} = L_\infty(w^n - w_0^n) = \max_{1 \le i \le N_P}(|w_i^n - w_{0,i}^n|)$$

$$\text{rmsErr} = L_2(w^n - w_0^n) = \sqrt{\frac{1}{N_P}\sum_{i=1}^{N_P}(w_i^n - w_{0,i}^n)^2}$$

An optional option –patches will be added to calculate the L_∞ and L_2 norm of the error in patch-internal-field (cells that directly connecting the patch) for specified patches.

NavyFOAMToTecplot

Description

This utility reads the scalar and vector volume and boundary patch fields from FOAM numerical solution data files and converts them to Tecplot data files.

This utility is part of NavyFOAM.

Usage

The command line usage looks like

```
NavyFoamToTecplot [-region name] [-case dir] [-fields fieldsList]
 [-patches patchesList] [-time time] [-latestTime]
```

The optional options are

-region name: specifies the region name;

-case dir: specifies the case directory;

-fields fieldsList: specifies a list of fields to output, e.g. '(p U gamma)', or '(U)';

-patches patchesList: specifies a list of patches to output, e.g. '(inlet outlet wall)', or '(hull)';

-time time: selects the time step;

-latestTime: selects the latest time step.

Without any option, the utility will read the data files for all time steps stored under the current case directory and convert the volume field to Tecplot data.

Installation

1) Create a working copy using svn checkout. The recommended local directory to checkout the package is

NavyFOAM/applications/utilities/postProcessing/dataConversion

```
svn checkout svn+ssh://blackwater/5700/NavyFOAM/IntegrationBranches/NavyFOAM-1.5-dev-rev1745/NavyFOAM/applications/utilities/postProcessing/dataConversion
/NavyFoamToTecplot
```

2) Go to directory ./dataConversion/NavyFoamToTecplot and compile the package:

```
wmake
```

The generated executable file NavyFoamToTecplot can be found in a user application binary file directory specified by $FOAM_USER_APPBIN.

Output

The output files can be found in $CASE_DIR/TecplotData, for example

```
NavyFoamToTecplot -fields '(p U)' -patches '(hull)' -time 10
```

will generate 10.dat and hull_10.dat in $CASE_DIR/TecplotData.

NavyCellSetToTecplot

Description

This utility reads a user specified *cellSet* file in ./constant/polymesh/sets/ and convert it to Tecplot data file.

This utility is part of NavyFOAM.

Usage

The command line usage looks like

```
NavyCellSetToTecplot <cellSetFileName> [-region name] [-case dir]
[-patches patchesList]
```

The mandatory argument is
cellSetFileName: specifies the file name for the cellSet, e.g. waveDampingCells;

The optional options are

-region name: specifies the region name;

-case dir: specifies the case directory;

-patches patchesList: specifies a list of patches to output, e.g. '(inlet outlet wall)', or '(hull)';

Installation

1) Create a working copy using svn checkout. The recommended local directory to checkout the package is

NavyFOAM/applications/utilities/postProcessing/dataConversion

```
svn checkout svn+ssh://blackwater/5700/NavyFOAM/IntegrationBranches/NavyFOAM-1.5-dev-rev1745/NavyFOAM/applications/utilities/postProcessing/dataConversion
/NavyCellSetToTecplot
```

2) Go to directory ./dataConversion/NavyCellSetToTecplot and compile the package:

```
wmake
```

The generated executable file NavyCellSetToTecplot can be found in a user application binary file directory specified by $FOAM_USER_APPBIN.

Output

The output files can be found in $CASE_DIR/TecplotData, for example

```
NavyCellSetToTecplot waveDampingCells -patches '(hull symmetry)'
```

may generate waveDampingCells.dat in $CASE_DIR/TecplotData. The Tecplot data file waveDampingCells.dat should contain the following zones:

```
ZONE T = volMesh
...
ZONE T = hull
...
ZONE T = symmetry
...
ZONE T = waveDampingCells
```

The first zone is the volume mesh. The next two zones are the surface mesh for the user specified patches. The last zone is a volume mesh for the cellSet. In Tecplot, the mesh of each zone can be plotted in different colors.

NavyFaceSetToTecplot

Description

This utility reads a user specified *faceSet* file in ./constant/polymesh/sets/ and converts it to Tecplot data file.

This utility is part of NavyFOAM.

Usage

The command line usage looks like

```
NavyFaceSetToTecplot <faceSetFileName> [-region name] [-case dir]
 [-patches patchesList]
```

The mandatory argument is

faceSetFileName: specifies the file name for the faceSet, e.g. skewFaces;

The optional options are
-region name: specifies the region name;
-case dir: specifies the case directory;
-patches patchesList: specifies a list of patches to output, e.g. '(inlet outlet wall)', or '(hull)';

Installation

1) Create a working copy using svn checkout. The recommended local directory to checkout the package is

NavyFOAM/applications/utilities/postProcessing/dataConversion

```
svn checkout svn+ssh://blackwater/5700/NavyFOAM/IntegrationBranches/NavyFOAM-1.5-dev-
rev1745/NavyFOAM/applications/utilities/postProcessing/dataConversion
/NavyFaceSetToTecplot
```

2) Go to directory ./dataConversion/NavyFaceSetToTecplot and compile the package:

```
wmake
```

The generated executable file NavyFaceSetToTecplot can be found in a user application binary file directory specified by $FOAM_USER_APPBIN.

Output

The output files can be found in $CASE_DIR/TecplotData, for example

```
NavyFaceSetToTecplot skewFaces -patches '(hull symmetry)'
```

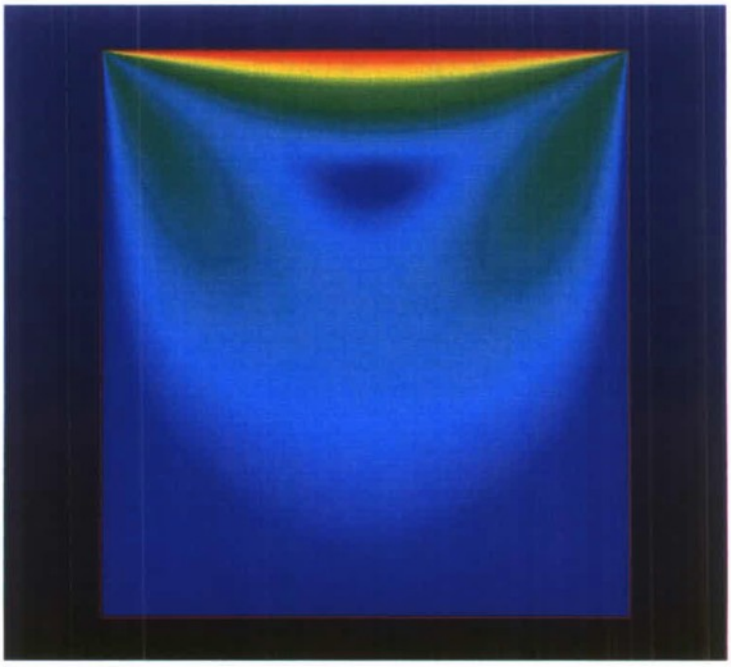
may generate skewFaces.dat in $CASE_DIR/TecplotData. The Tecplot data file skewFaces.dat should contain the following zones:

```
ZONE T = volMesh
...
ZONE T = hull
...
ZONE T = symmetry
...
ZONE T = skewFaces
```

The first zone is the volume mesh. The next two zones are the surface mesh for the user specified patches. The last zone is a surface mesh for the faceSet. In Tecplot, the mesh of each zone can be plotted in different colors.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix C: icoFoam Lid Driven Cavity Tutorial



This tutorial involves using the laminar, transient, incompressible solver for a 2-D cavity. The cavity consists of 4 walls, where the top wall is moving, and the other walls are stationary. First, we will go over pre-processing and case setup, then we will run the test case, and finally we will look at some post-processed results.

For more detailed information on the OpenFOAM code and settings consult the User's Guide: http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf.

Pre-Processing and Case Setup

Upon looking in the case directory (screen capture below) you will notice that there are directories labeled ***system*** and ***0***, and a file named ***transportProperties***. There is also a file labeled ***2D_cavity_allCoarseStr.cas***, which is a mesh exported from Gridgen in Fluent ASCII double precision format. We will discuss the existing files and directories later, but now we need to import the fluent .cas file into OpenFoam. This will be done using the command ***fluentMeshToFoam***.

```
[kdelaney@Retech allCoarseStruct]$ l
total 1.2M
-rw-rw-r--  1 kdelaney kdelaney 1.2M Feb 19 12:31 2D_cavity_allCoarseStr.cas
drwxrwxr-x  2 kdelaney kdelaney 4.0K Mar  9 16:53 system
-rw-r-----  1 kdelaney kdelaney  886 Apr  1 08:35 transportProperties
drwxr-xr-x  2 kdelaney kdelaney 4.0K Apr  1 08:35 0
```

Mesh Input

Now enter the ***fluentMeshToFoam*** command as seen below and view the output from the screen dump. The extra "| tee conversionLog" is not essential and is only included to record

the screen dump in a file named *conversionLog*. Your output should be the same as the screen captures.

There is a lot of information on the screen dump, most of which is self explanatory. The most important part to notice is the last two lines of text in the screen dump which tell you that the mesh information has been written into a directory named ***polyMesh*** inside a newly created directory named ***constant***. Finally the command ends successfully with the "End." statement.

```
[kdelaney@Retech allCoarseStruct]$ fluentMeshToFoam 2D_cavity_allCoarseStr.cas | tee conversionLog
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Exec   : fluentMeshToFoam 2D_cavity_allCoarseStr.cas
Date   : Mar 09 2010
Time   : 16:40:45
Host   : Retech
PID    : 5119
Case   : /home/kdelaney/NavyFoam_runs/run/flat_plate/movingWallCavity/allCoarseStruct
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Reading header: "exported from Gridgen 15.14R1"
Dimension of grid: 3
Number of points: 9522
number of faces: 18632
Number of cells: 4624
Reading points
Other readCellGroupData: 2 1 1210 1 4
Reading uniform cells
Read zone1:2 name:fluid patchTypeID:fluid
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:3 name:interior-3 patchTypeID:interior
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:4 name:Inlet-4 patchTypeID:wall
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:5 name:Top-5 patchTypeID:wall
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:6 name:side_1-6 patchTypeID:wall
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:7 name:Outlet-7 patchTypeID:wall
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
Read zone1:8 name:Bottom-8 patchTypeID:wall
Reading zone data
Embedded blocks in comment or unknown: (
Found end of section in unknown:)
,Reading uniform faces
```

(screen output continues on the next page…)

```
Read zone1:9 name:side_2-9 patchTypeID:wall
Reading zone data


FINISHED LEXING


dimension of grid: 3
Creating shapes for 3-D cells
Building patch-less mesh...--> FOAM Warning :
    From function polyMesh::polyMesh(... construct from shapes...)
    in file meshes/polyMesh/polyMeshFromShapeMesh.C at line 581
    Found 9520 undefined faces in mesh; adding to default patch.
done.

Building boundary and internal patches.
Creating patch 0 for zone: 3 start: 1 end: 9112 type: interior name: interior-3
Creating patch 1 for zone: 4 start: 9113 end: 9180 type: wall name: Inlet-4
Creating patch 2 for zone: 5 start: 9181 end: 9248 type: wall name: Top-5
Creating patch 3 for zone: 6 start: 9249 end: 13872 type: wall name: side_1-6
Creating patch 4 for zone: 7 start: 13873 end: 13940 type: wall name: Outlet-7
Creating patch 5 for zone: 8 start: 13941 end: 14008 type: wall name: Bottom-8
Creating patch 6 for zone: 9 start: 14009 end: 18632 type: wall name: side_2-9
Patch interior-3 is internal to the mesh  and is not being added to the boundary.
Adding new patch Inlet-4 of type wall as patch 0
Adding new patch Top-5 of type wall as patch 1
Adding new patch side_1-6 of type wall as patch 2
Adding new patch Outlet-7 of type wall as patch 3
Adding new patch Bottom-8 of type wall as patch 4
Adding new patch side_2-9 of type wall as patch 5

Default patch type set to empty

Writing mesh... to "constant/polyMesh"  done.


End
```

constant/ directory and the createPatch Command

All of the mesh geometry details are stored in the constant/polyMesh directory. The boundary file is typically the only one in polyMesh directory that gets edited.

Now open up your constant/polyMesh/boundary file, which contains all of the information for surfaces that were imported from your mesh. It should look like the screen capture seen below.

The information at the top of the file (under the FoamFile header) gives a description of the file (version, format, class, etc.) and is most likely only useful to more experienced users, but at the very least it is always useful as a label to let you know where you are. Further down the file you can see that 6 surfaces (Inlet-4, Top-5, etc.) were imported with the same boundary names that were created in Gridgen. Each surface is described by the type of OpenFoam surface, nFaces and startFace. Only the type is of concern to the user at this stage and that will be discussed in more detail later on.

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

6
(
    Inlet-4
    {
        type            wall;
        nFaces          68;
        startFace       9112;
    }
    Top-5
    {
        type            wall;
        nFaces          68;
        startFace       9180;
    }
    side_1-6
    {
        type            wall;
        nFaces          4624;
        startFace       9248;
    }
    Outlet-7
    {
        type            wall;
        nFaces          68;
        startFace       13872;
    }
    Bottom-8
    {
        type            wall;
        nFaces          68;
        startFace       13940;
    }
    side_2-9
    {
        type            wall;
        nFaces          4624;
        startFace       14008;
    }
)

// ************************************************************************* //
~
~
```

Often times the ***boundary*** file needs to be altered from what is originally created during the import process. For this case we need to edit the ***type*** of surface for some surfaces, and we would like to group certain surfaces together to avoid redundancy and make life easier.

First we will group some of the surfaces together for case of book-keeping. To group surfaces together, we use the ***createPatch*** command. For now let's say that we want to create a group of surfaces for what will be: the moving lid (***movingWall***), the stationary walls (***fixedWalls***), and the 2-D surfaces on the front and back (***frontAndBack***). The file that allows us to group surfaces is called ***createPatchDict*** and it is located in the ***system*** directory.

If you open your ***system/createPatchDict*** file you will notice that it needs to be edited. Again, there is detailed information about this dictionary file underneath the ***FoamFile*** header, and can be used as a reference to the user. Underneath the ***FoamFile*** section are the ***matchToTolerance*** and ***pointSync*** commands which are not important right now and should be left as is. Next, you will see a ***patches*** section, which is where we will do our editing to join surfaces under one boundary name and ***type***.

The user should edit their ***createPatchDict*** file ***patches()*** section to look like the screen capture on the next page. The first patch is essentially renaming the ***Top-5*** boundary to ***movingWall***, while leaving the type as ***wall***. This is equivalent to simply changing the name in the original ***constant/polyMesh/boundary*** file, but was done here for educational purposes. The second patch will group the ***Inlet-4, Bottom-8,*** and ***Outlet-7*** surfaces into one boundary named ***fixedWalls***, and this new boundary will remain a type ***wall***. Lastly, the ***side_1-6*** and ***side_2-9*** surfaces will be combined to ***frontAndBack*** and this new boundary will be of type ***empty***. The ***empty*** patch type is required for ALL 2-D surfaces.

Now we are ready to combine the surfaces, but first it is generally a good idea to copy our ***constant*** directory before we combine our surfaces, so there is always a reference. So for Linux users, simply:

```
>> cp -r constant orig_constant
```

to copy our reference ***constant*** folder to ***orig_constant***. Then in the case directory enter "***createPatch***" in the command line. The resulting screen dump should look like the following screen capture.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.0                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "/home/penfold/mattijs/foam/mattijs2.1/run/icoFoam";
    case            "cavity";
    instance        "system";
    local           "";

    class           dictionary;
    object          createPatcheDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Tolerance used in matching faces. Absolute tolerance is span of
// face times this factor.
matchTolerance 1E-3;

// Do a synchronisation of coupled points.
pointSync true;


// Patches to create.
// If no patches does a coupled point and face synchronisation anyway.
patches
(


    {
        name movingWall;
        type wall;
        constructFrom patches;
        patches ( Top-5 );
    }

    {
        name fixedWalls;
        type wall;
        constructFrom patches;
        patches ( Inlet-4 Bottom-8 Outlet-7 );
    }

    {
        name frontAndBack;
        type empty;
        constructFrom patches;
        patches (side_1-6 side_2-9 );
    }

);
```

```
[kdelaney@Retech allCoarseStruct]$ cp -r constant/ orig_constant
[kdelaney@Retech allCoarseStruct]$
[kdelaney@Retech allCoarseStruct]$
[kdelaney@Retech allCoarseStruct]$
[kdelaney@Retech allCoarseStruct]$
[kdelaney@Retech allCoarseStruct]$ createPatch
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Exec   : createPatch
Date   : Mar 09 2010
Time   : 16:53:16
Host   : Retech
PID    : 5158
Case   : /home/kdelaney/NavyFoam_runs/run/flat_plate/movingWallCavity/allCoarseStruct
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Reading createPatchDict.

Using relative tolerance 0.001 to match up faces and points

Create polyMesh for time = 0

Adding new patch movingWall of type wall as patch 6
Adding new patch fixedWalls of type wall as patch 7
Adding new patch frontAndBack of type empty as patch 8

Moving faces from patch Top-5 to patch 6
Moving faces from patch Inlet-4 to patch 7
Moving faces from patch Bottom-8 to patch 7
Moving faces from patch Outlet-7 to patch 7
Moving faces from patch side_1-6 to patch 8
Moving faces from patch side_2-9 to patch 8

Doing topology modification to order faces.

Synchronising points.
Points changed by average:0 max:0

Removing patches with no faces in them.

Removing empty patch Inlet-4 at position 0
Removing empty patch Top-5 at position 1
Removing empty patch side_1-6 at position 2
Removing empty patch Outlet-7 at position 3
Removing empty patch Bottom-8 at position 4
Removing empty patch side_2-9 at position 5
Removing patches.
Writing repatched mesh to 0.005

End
```

You will notice that the second to last line states "Writing repatched mesh to 0.005". The ***createPatch*** command will write the new patched surface information into a directory whose name is the first time step output of your future run, which in this case is a ***0.005***.

So now you need to get rid of the old ***constant*** directory and move the new ***0.005*** directory to ***constant.*** In Linux this would be accomplished by:

>> mv constant orig_constant

>> mv 0.005 constant

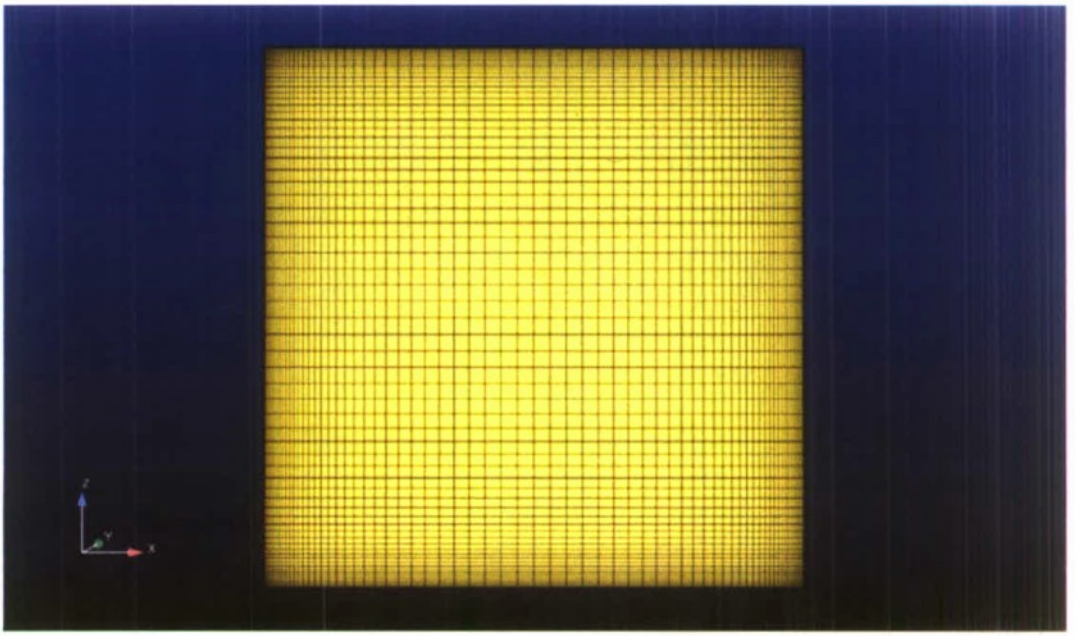
Now the ***transportProperties*** file needs to be placed in the ***constant*** directory. The ***transportProperties*** file must ALWAYS be present in the ***constant*** directory. In Linux this would be accomplished by:

>> mv transportProperties constant/

Now if you open the ***constant/polyMesh/boundary*** file it should have the correct number of patches, names, and types. Your new ***boundary*** file should look like the screen capture on the next page.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

3
(
    movingWall
    {
        type            wall;
        nFaces          68;
        startFace       9112;
    }
    fixedWalls
    {
        type            wall;
        nFaces          204;
        startFace       9180;
    }
    frontAndBack
    {
        type            empty;
        nFaces          9248;
        startFace       9384;
    }
)

// ************************************************************************* //
```

Now we have the geometry imported and named as we want for the run. A good next step is to export the geometry into a visual package (EnSight, ParaView, etc.) and make sure that all surfaces are grouped and labeled correctly. To export the geometry, use ***foamToEnsight*** for EnSight, ***foamToVTK*** for ParaView, and no additional command is needed for ParaFoam. So now take a minute or two and inspect your geometry in your package of choice. Your geometry should look like the below figure, with the appropriate surface labels.



Cavity geometry as seen in EnSight

Material Properties

The next step is to set up the material properties for the fluid. For the ***icoFoam*** solver, only the kinematic viscosity is required in the ***constant/transportProperties*** dictionary file. Open the ***transportProperties*** file, it should look like the sereen capture below. No editing is necessary, just note that kinematie viseosity is always set in ***transportProperties***.

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              nu [0 2 -1 0 0 0 0] 0.01;

// ************************************************************************* //
```

The kinematic viscosity, *nu*, is entered in as:

nu nu [0 2 -1 0 0 0] 0.01;

where *[0 2 -1 0 0 0 0]* sets the units based on [Mass Length Time Temperature Quantity Current Luminous intensity]. The kinematic viscosity dimensions are Length^2/Time or in SI units: m^2/s. The value of the kinematic viscosity is set to 0.01. Remember that this value must always be consistent with the Reynolds number,

Re = UL/nu

0/ directory (Initial and Boundary Conditions)

Now we turn our attention to the initial and boundary conditions, which are stored in the *0/* directory.

For the ***icoFoam*** solver only ***U*** and ***p*** files are needed in the *0/* directory.

Open the ***0/U*** file; it should look like the screen capture below.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }

    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    frontAndBack
    {
        type            empty;
    }
}

// ************************************************************************* //
~
```

In the ***0/U*** file notice:

U is a vector field quantity. All ***U*** values must be set in vector format, ***(X X X)***.

The *U* dimensions must match the variable by M,L,T,… so velocity is *[0 1 -1 0 0 0 0]*

The "internalField" sets the initial flow field condition for U. For this case the fluid is initially at rest (Ux,Uy,Uz = 0)

The "boundaryField" sets velocity boundary conditions for ALL surfaces. All surfaces must be included with proper BC's. All surface names must match the ***constant/polyMesh/boundary*** surface name exactly. The order of surfaces is not important, but the names must match identically.

The velocity boundary conditions for the three surfaces are as follows:

The ***movingWall*** is set with

type fixedValue;

value uniform (1 0 0);

for the top lid to move with velocity of ***Ux***=1.

The ***fixedWalls*** are set with

type fixedValue;

value uniform (0 0 0);

to apply the no-slip boundary condition to the walls.

The ***frontAndBack*** surfaces are set with

type empty;

because they are the 2-D boundaries. ALL 2-D boundaries must have ***type*** set to ***empty***.

Again, the order of the surfaces in the ***0/...*** files doesn't matter, but the names and ***types*** MUST be consistent with those listed in the ***constant/polyMesh/boundary*** file.

Now open the ***0/p*** file, it should look like the screen capture below.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    movingWall
    {
        type            zeroGradient;
    }

    fixedWalls
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            empty;
    }

}

// ************************************************************************* //
```

In the ***0/p*** file notice:

p is a scalar field quantity. All ***p*** values must be set as a sealar, ***X***, value.

The ***p*** dimensions must mateh the variable by M,L,T,... so pressure is [0 2 -2 0 0 0 0], becausc in icoFoam ***p*** is actually the pressure dividcd by the density, thus the SI units would bc m^2/s^2.

The "internalField" sets the initial eondition for ***p***. For this case (and most othcrs) we do not care about the absolute value of thc pressure, ***p***, so we just set it to 0 for easc.

The ***boundaryField*** sets pressure boundary conditions for ALL surfaces. All surfaces must be included with proper BC's that are consistent with the ***constant/polyMesh/boundary*** surfaces.

The pressure boundary conditions for the three surfaces are as follows:

The ***movingWall*** is set with

type zeroGradient;

for the moving wall.

The ***fixedWalls*** are set with

type zeroGradient;

for the no-slip wall.

The ***frontAndBack*** surfaces are set with

type empty;

because they are the 2-D boundaries. ALL 2-D boundaries must have ***type*** set to ***empty***.

system/ directory (Solver Settings)

Now we will look at some of the solver settings and controls that are located in the ***system/*** directory. We will focus on the ***controlDict***, ***fvSolution***, and ***fvSchemes*** files. We already used the ***createPatchDict*** to merge multiple surfaces.

Open the ***system/controlDict*** dictionary file. It should look like the screen capture below. The ***controlDict*** file sets all of the run-time parameters and output information. This is also where run-time libraries and functions, such as force outputs over a patch and dynamic mesh libraries are specified.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.3;

deltaT          0.00001;

writeControl    timeStep;

writeInterval   3000;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

// ************************************************************************* //
~
~
~
```

The solver specified in ***application*** input does not matter. The solver is specified on the command line or in a script file. Thus, this is an insignificant line for our purposes.

The solver settings are fairly obvious, and more detail is provided on page U-108 of the User's Guide (http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf). For now we will only cover a broad view of the file.

We know that ***icoFoam*** is a transient solver. We see that we will run the simulation from 0 to 0.3 seconds in time steps of 0.00001 seconds with the solver writing output information every 3000 time steps (t=0.03, 0.06, 0.09, ...). The data will be written in ASCII format in directories that are denoted by time 6 digits long. Notice that ***runTimeModifiable*** is chosen to ***yes***, this means that we can make changes to the ***controlDict*** in the middle of a run, and they will be adjusted on the fly, as opposed to having the settings set in stone for the whole calculation.

One important note is that to start a calculation from a previous solution the ***startFrom*** entry must be switched to ***latestTime***, and desired start time information (directory and BC's) must be present in the case directory.

Now open the ***system/fvSolution*** dictionary file. It should look like the screen capture on the next page.

The ***fvSolution*** file contains linear solver information as well as solver algorithm settings.

The ***solvers*** section contains linear solver settings for pressure and velocity. Note that for this case we are using preconditioned conjugate gradient solvers (***PCG*** for symmetric matrices and ***PBiCG*** for asymmetric matrices), but we also commonly use multi-grid solvers (GAMG, AAMG, etc.). The solver tolerance and relative tolerance settings are not important right now. The ***minIter*** command sets a minimum number of times the linear solver will iterate on a variable. It is usually recommended that the user always set a minimum number of iterations > 0 to prevent the solver from prematurely not solving for a variable.

Below the ***solvers*** section are pressure-implicit split-operator (***PISO***) algorithm control settings. These ***PISO*** settings are not particularly useful to the user at this time, so only a broad view of what each setting means is given. Also, note that the ***PISO*** algorithm must be used for all transient solvers and the ***SIMPLE*** algorithm must be used for all steady-state solvers. For this case we have ***nCorrectors*** set to 2, which means that we will solve the pressure equation twice per time iteration. The value of ***nNonOrthogonalCorrectors*** is set to 0. This parameter is not particularly important to the user at this moment. Notice that we have set cell number 0 as our reference cell, where the reference value is 0. This is the reference pressure for the incompressible solver.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
        minIter          5;
    };

    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
        minIter          5;
    };
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

// ************************************************************************* //
~
```

Now open the ***system/fvSchemes*** dictionary file. It should look like the screen capture below.

Many of the ***fvSchemes*** settings are not particularly useful to the user at this time, so only a broad view of what each setting means is given. For more detail on these settings consult page U-110 of the User's Guide. More detail of the discretization settings is given in the ***simpleFoam*** and ***rasInterFoam*** tutorials.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}

laplacianSchemes
{
    default         none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p;
}

// ************************************************************************* //
```

The ***fvSchemes*** file sections declares the following settings:

ddt → time discretization

gradSchemes → gradient term discretizations

divSchemes → divergence terms discretization

laplacianSchemes → Laplacian terms discretization

interpolationSchemes → interpolation of values from cell centers to cell face centers

snGradSchemes → surface normal gradient evaluation at cell faces

fluxRequired → lists fields where flux is generated in the application

For all of the ***fvSchemes*** fields a ***default*** value can be specified, or ***default*** can be set to ***none*** which means that the user must enter all values for the appropriate variables themselves.

Running icoFoam

Now we are ready to run. Type ***icoFoam*** on the command line like the screen capture below, and hit "*ctrl+c*" to take a look at the first few iterations (piping the screen dump to a log file by using "*icoFoam | tee log*" is another option).

```
[kdelaney@Retech tut_allCoarseStruct]$ icoFoam
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Exec   : icoFoam
Date   : Jun 30 2010
Time   : 14:53:06
Host   : Retech
PID    : 20508
Case   : /home/kdelaney/NavyFoam_runs/run/flat_plate/movingWallCavity/tut_allCoarseStruct
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

Reading transportProperties

Reading field p

Reading field U

Reading/calculating face flux field phi


Starting time loop

Time = 1e-05

Courant Number mean: 0 max: 0 velocity magnitude: 0
PBiCG:  Solving for Ux, Initial residual = 1, Final residual = 1.68372e-20, No Iterations 5
PBiCG:  Solving for Uz:  solution singularity
PCG:  Solving for p, Initial residual = 1, Final residual = 6.77855e-07, No Iterations 138
time step continuity errors : sum local = 1.14833e-15, global = 3.92084e-26, cumulative = 3.92084e-26
PCG:  Solving for p, Initial residual = 0.0407237, Final residual = 8.83925e-07, No Iterations 126
time step continuity errors : sum local = 3.57796e-15, global = -5.09959e-25, cumulative = -4.7075e-25
ExecutionTime = 0.13 s  ClockTime = 0 s

Time = 2e-05

Courant Number mean: 1.36506e-07 max: 9.7988e-05 velocity magnitude: 0.0362317
PBiCG:  Solving for Ux, Initial residual = 0.134603, Final residual = 1.07427e-21, No Iterations 5
PBiCG:  Solving for Uz, Initial residual = 0.333103, Final residual = 8.02106e-21, No Iterations 5
PCG:  Solving for p, Initial residual = 0.100492, Final residual = 8.49585e-07, No Iterations 125
time step continuity errors : sum local = 3.78654e-15, global = 3.849e-25, cumulative = -8.58504e-26
PCG:  Solving for p, Initial residual = 0.0020439, Final residual = 9.96011e-07, No Iterations 112
time step continuity errors : sum local = 4.8709e-15, global = 4.60384e-25, cumulative = 3.74534e-25
ExecutionTime = 0.2 s  ClockTime = 0 s

Time = 3e-05

Courant Number mean: 2.66818e-07 max: 0.000190662 velocity magnitude: 0.0705446
PBiCG:  Solving for Ux, Initial residual = 0.0711115, Final residual = 5.11184e-22, No Iterations 5
PBiCG:  Solving for Uz, Initial residual = 0.196949, Final residual = 5.20581e-21, No Iterations 5
```

Some observations from the first few iterations:

You can see that the solver started from time equal to 0 seconds and is marching in inerements of 1e-5 seconds.

For each iteration the pressure equation is solved twice and the velocity equations are solved once. For each variable linear solver we can see the initial residual, final residual, and the number of iterations it took to drop from the initial to the final residual. We set all of these toleranees and iteration eriteria in the ***system/fvSolution*** dietionary file.

There are also Courant number and eontinuity error reports.

The best way to typically monitor the solution is to make sure that the velocity magnitude stays at a reasonable number, and make sure that initial pressure residuals are decreasing or are holding steady at an acceptable value.

The last line of the time iteration produces execution and clock time information. This is useful in gauging the efficiency of your solution.

Now let the *icoFoam* solver go until 0.5 seconds to get a converged solution.

Post-Processing

Notice that there are many time directories in your case directory. Each of these directories contains output information for their respective time step.

To look at the post-processed results simply type the following commands, depending on the post-processing tool of choice:

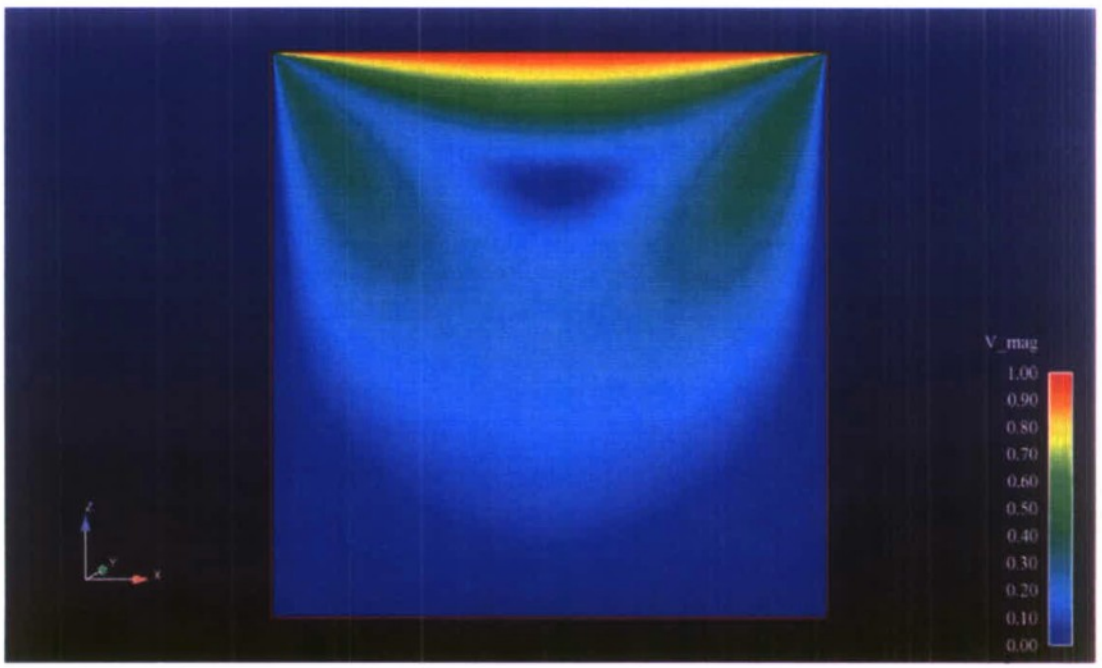
>> foamToEnSight -latestTime → to look at the results in EnSight

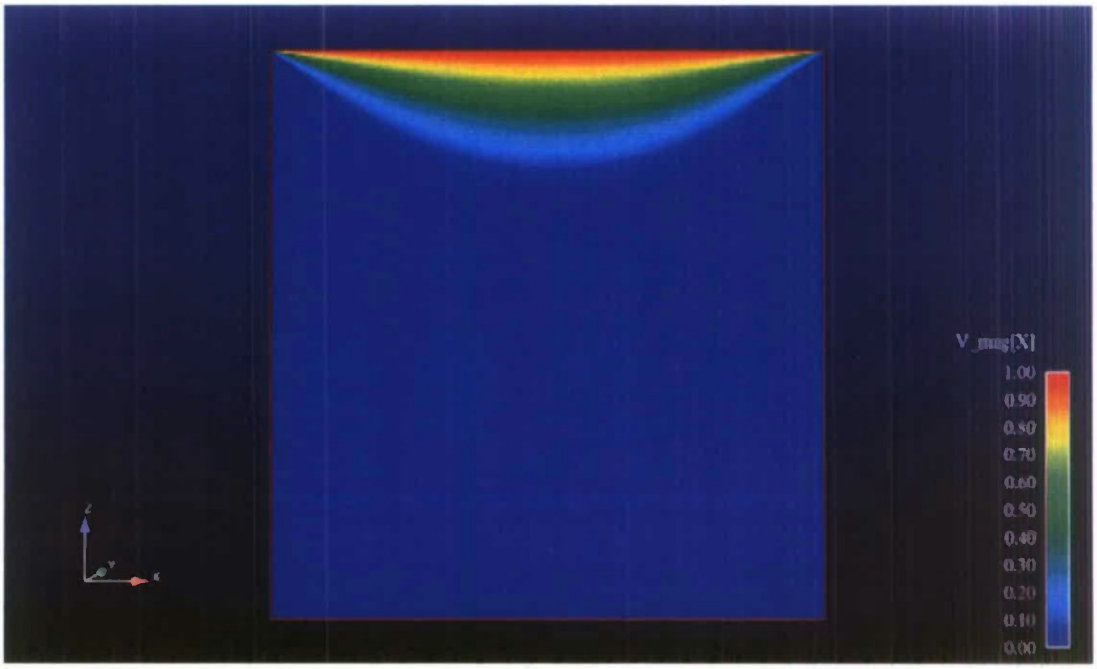
>> foamToVTK -latestTime → to look at the results in ParaView

where the command *–latestTime* is used to only look at the results from the last output time step. To look at the results for all time steps simply leave off the *–latestTime* command, and to look at the results for a specific time (ie 0.005) use *–time 0.005*.

To look at the results in ParaFoam, no additional commands are needed, simply open ParaFoam in the case directory.

Your results should look like the velocity magnitude (*Vmag*) and axial velocity (*Vmag[x]*) contours below.




V_mag[X]
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00

Appendix D: simpleFOAM Body-1 Tutorial

This tutorial involves using the turbulent, steady, ineompressible solver for a 3-D body-of-revolution, the Body-1. Only half the body is solved, as symmetry is assumed. The domain is non-dimensionalized by length, so all lengths in the domain are normalized by body length. First, we will go over pre-processing and case setup, then we will run the test case, and finally we will look at some post-processed results.

For more detailed information on the OpenFOAM code and settings consult the User's Guide: http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf.

Pre-Processing and Case Setup

Upon looking in the ease directory (sereen eapture below) you will notice that there are directories labeled ***system*** and ***0***, and a file named ***transportProperties***. There is a parallel processing script named ***oFOAM.scp***. There is also a file labeled ***body1_Box_ASCII.fluent.cas***, which is a mesh exported from Gridgen in Fluent ASCII double precision format. We will discuss the existing files and direetories later, but now we need to import the fluent .cas file into OpenFoam. This will be done using the command ***fluentMeshToFoam***.

```
[delaneyk@amazon body1_Tutorial]$ l
total 237M
-rw-r--r--  1 delaneyk users 1.6K Apr  8 15:17 transportProperties
-rwxr-xr-x  1 delaneyk users 4.6K Apr  8 15:17 RASProperties
drwxr-xr-x  2 delaneyk users   80 Apr  8 15:17 system
drwxr-xr--  2 delaneyk users   81 Apr  8 15:17 0
-rw-r--r--  1 delaneyk users  658 Apr  8 15:17 oFOAM.scp
-rw-r--r--  1 delaneyk users 237M Apr  8 15:31 body1_Box-ASCII.fluent.cas
```

Mesh Input

Now enter the ***fluentMeshToFoam*** eommand as seen below and view the output from the screen dump. Your output should be the same as the screen captures on the next pages.

There is a lot of information on the screen dump, most of which is self explanatory. The most important part to notice is the last two lines of text in the screen dump which tell you that the mesh information has been written into a directory named ***polyMesh*** inside a newly created directory named ***constant***. Finally the command ends successfully with the "End." statement.

```
[delaneyk@amazon body1_Tutorial]$ fluentMeshToFoam body1_Box-ASCII.fluent.cas
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : fluentMeshToFoam body1_Box-ASCII.fluent.cas
Date   : Apr 08 2010
Time   : 15:31:37
Host   : amazon.dt.navy.mil
PID    : 19865
Case   : /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/delaneyk-1.5-dev/run/body1/boundingBoxFar
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPOpt/li
Model11printCoeffsEv
Dimension of grid: 3
@@Number of cells: 2180547
number of faces: 5106844
Number of points: 860613
Other readCellGroupData: a 1 2145c3 1 0
Reading mixed cells
Reading mixed faces
Reading uniform faces
Reading uniform faces
Reading uniform faces
Reading uniform faces
Reading uniform faces
Reading uniform faces
Reading mixed faces
Reading points
Read zone2:10 name:fluid patchTypeID:fluid
Reading zone data

Read zone2:8 name:symmetry patchTypeID:symmetry
Reading zone data

Read zone2:6 name:outlet patchTypeID:pressure-outlet
Reading zone data

Read zone2:5 name:inlet patchTypeID:inlet-vent
Reading zone data

Read zone2:4 name:farfield patchTypeID:pressure-far-field
Reading zone data

Read zone2:1 name:bow patchTypeID:wall
Reading zone data

Read zone2:2 name:midbody patchTypeID:wall
Reading zone data

Read zone2:3 name:stern patchTypeID:wall
```

(screen output continues on the next page...)

```
Reading zone data

Read zone2:9 name:interior-faces patchTypeID:interior
Reading zone data



FINISHED LEXING


dimension of grid: 3
Creating shapes for 3-D cells
Building patch-less mesh...--> FOAM Warning :
    From function polyMesh::polyMesh(... construct from shapes...)
    in file meshes/polyMesh/polyMeshFromShapeMesh.C at line 581
    Found 111916 undefined faces in mesh; adding to default patch.
done.

Building boundary and internal patches.
maxZoneID: 9
Creating patch 0 for zone: 9 start: 1 end: 4994928 type: interior name: interior-faces
Creating patch 1 for zone: 1 start: 4994929 end: 5001640 type: wall name: bow
Creating patch 2 for zone: 2 start: 5001641 end: 5013072 type: wall name: midbody
Creating patch 3 for zone: 3 start: 5013073 end: 5035504 type: wall name: stern
Creating patch 4 for zone: 4 start: 5035505 end: 5037282 type: pressure-far-field name: farfi
Creating patch 5 for zone: 5 start: 5037283 end: 5037724 type: inlet-vent name: inlet
Creating patch 6 for zone: 6 start: 5037725 end: 5038166 type: pressure-outlet name: outlet
Creating patch 7 for zone: 8 start: 5038167 end: 5106844 type: symmetry name: symmetry
Patch interior-faces is internal to the mesh  and is not being added to the boundary.
Adding new patch bow of type wall as patch 0
Adding new patch midbody of type wall as patch 1
Adding new patch stern of type wall as patch 2
Adding new patch farfield of type patch as patch 3
Adding new patch inlet of type patch as patch 4
Adding new patch outlet of type patch as patch 5
Adding new patch symmetry of type symmetryPlane as patch 6

Default patch type set to empty

Writing mesh... to "constant/polyMesh"  done.


End
```

Now run the ***checkMesh*** command for two reasons:

- to make sure the mesh was imported correctly
- to asses the quality of the mesh for the OpenFOAM solver

Your ***checkMesh*** output should look like the screen captures on the next pages.

```
[delaneyk@amazon body1_Tutorial]$ checkMesh
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : checkMesh
Date   : Apr 08 2010
Time   : 15:36:20
Host   : amazon.dt.navy.mil
PID    : 21145
Case   : /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/delaneyk-1.5-dev/run/body1/boundingBoxFar
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPOpt/li
Model11printCoeffsEv
Create polyMesh for time = constant

Time = constant

Mesh stats
    points:           860613
    faces:            5106844
    internal faces:   4994928
    cells:            2180547
    boundary patches: 7
    point zones:      0
    face zones:       0
    cell zones:       0

Number of cells of each type:
    hexahedra:     0
    prisms:        1379584
    wedges:        0
    pyramids:      0
    tet wedges:    0
    tetrahedra:    800963
    polyhedra:     0

Checking topology...
    Boundary definition OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).

Checking patch topology for multiply connected surfaces ...
    Patch               Faces    Points   Surface topology
    bow                 6712     3459     ok (non-closed singly connected)
    midbody             11432    5882     ok (non-closed singly connected)
    stern               22432    11490    ok (non-closed singly connected)
    farfield            1778     950      ok (non-closed singly connected)
    inlet               442      252      ok (non-closed singly connected)
    outlet              442      252      ok (non-closed singly connected)
```

(Screen capture continues on next page...)

```
    symmetry            68678    50479    ok (non-closed singly connected)

Checking geometry...
    This is a 3-D mesh
    Overall domain bounding box (-10 -10 0) (10 10 10)
    Mesh (non-empty) directions (1 1 1)
    Mesh (non-empty, non-wedge) dimensions 3
    Boundary openness (-1.71587e-18 -1.21515e-17 -5.8041e-16) Threshold = 1e-06 OK.
    Max cell openness = 5.13185e-14 OK.
    Max aspect ratio = 564.729 OK.
    Minumum face area = 5.88008e-10. Maximum face area = 1.24594.  Face area magnitudes OK.
    Min volume = 6.96628e-14. Max volume = 0.421459.  Total volume = 4000.  Cell volumes OK.
    Mesh non-orthogonality Max: 61.1137 average: 9.70308 Threshold = 70
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 1.25241 OK.

Mesh OK.

End
```

There is a lot (probably too much) of information with the ***checkMesh*** screen dump. At the top, the ***Mesh stats*** section shows that the mesh has 2.18 million total cells/elements, and the ***Number of cells of each type*** section shows 1.38 million arc prisms and 0.8 million are tetrahedral elements. Below that, we see that the topology checks out ***OK*** and that all the surfaces are correctly connected.

Finally, the ***Checking geometry...*** section displays mesh quality statistics. This section gives a lot of information, but the most important parts are the aspect ratio, non-orthogonality, and skewness. For this case all check out ***OK***, so we are free to proceed knowing the mesh is of high quality.

Sometimes it is not possible to create a mesh without any high aspect ratio, non-orthogonal, or skewed cells. In fact, most meshes created will contain bad cells, and run fine. However, at some point (which is not quantitatively clear) the mesh will be so poor it either won't run, or it will take a long time to run. There aren't exact guidelines on OpenFOAM mesh quality; it simply takes experience running various meshes.

constant/ directory and the createPatch Command

All of the mesh geometry details are stored in the ***constant/polyMesh*** directory. The ***boundary*** file is typically the only one in ***polyMesh*** directory that gets edited.

Now open up your ***constant/polyMesh/boundary*** file, which contains all of the information for surfaces that were imported from your mesh. It should look like the screen capture seen below.

The information at the top of the file (under the ***FoamFile*** header) gives a description of the file (version, format, class, etc.) and is most likely only useful to more experienced users, but at the very least it is always useful as a label to let you know where you are. Further down the file you can see that 7 surfaces (bow, midbody, etc.) were imported with the same boundary names that were created in ***Gridgen***. Each surface is described by the ***type*** of OpenFoam surface, ***nFaces*** and ***startFace***. Only the ***type*** is of concern to the user at this stage and that will be discussed in more detail later on.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

7
(
    bow
    {
        type            wall;
        nFaces          6712;
        startFace       4994928;
    }
    midbody
    {
        type            wall;
        nFaces          11432;
        startFace       5001640;
    }
    stern
    {
        type            wall;
        nFaces          22432;
        startFace       5013072;
    }
    farfield
    {
        type            patch;
        nFaces          1778;
        startFace       5035504;
    }
    inlet
    {
        type            patch;
        nFaces          442;
        startFace       5037282;
    }
    outlet
    {
        type            patch;
        nFaces          442;
        startFace       5037724;
    }
    symmetry
    {
        type            symmetryPlane;
        nFaces          68678;
        startFace       5038166;
    }
)
```

Often times the ***boundary*** file needs to be altered from what is originally created during the import process. For example, the hull might be imported as 5 different surfaces and you

would like to group them together as one surface. For this case we will group the separate hull surfaces together to avoid redundancy and make life easier.

To group separate surfaces together, we use the ***createPatch*** command. For now let's say that we want to create a group of surfaces for what will be the hull. The file that allows us to group surfaces is called ***createPatchDict*** and it is located in the ***system*** directory.

If you open your ***system/createPatchDict*** file you will notice that it needs to be edited to group the surfaces from ***constant/polyMesh/boundary***. Again, there is detailed information about this dictionary file underneath the ***FoamFile*** header, and can be used as a reference to the user. Underneath the ***FoamFile*** section are the ***matchToTolerance*** and ***pointSync*** commands which are not important right now and should be left as is. Next, you will see a ***patches*** section, which is where we will do our editing to join surfaces under one boundary name and ***type***.

The user should edit their ***createPatchDict*** file ***patches()*** section to look like the screen capture on the next page. The only patch will group the ***hull, midbody,*** and ***stern*** surfaces into one boundary named ***hull***, and this new boundary will remain a type ***wall***.

Now we are ready to combine the surfaces, but first it is generally a good idea to copy our ***constant*** directory before we combine our surfaces, so there is always a reference. So for Linux users, simply:

>> cp –r constant orig_constant

to copy our reference ***constant*** folder to ***orig_constant***. Then in the case directory enter "***createPatch***" in the command line. The resulting screen dump should look like the following screen capture.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.0                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "body 1";
    instance        "";
    local           "";

    class           dictionary;
    object          createPatcheDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Tolerance used in matching faces. Absolute tolerance is span of
// face times this factor.
matchTolerance 1E-3;

// Do a synchronisation of coupled points.
pointSync true;


// Patches to create.
// If no patches does a coupled point and face synchronisation anyway.
patches
(
    {
        name hull;

        type wall;

        constructFrom patches;

        patches (bow midbody stern);

    }
);

// ************************************************************************* //
```

```
[delaneyk@amazon body1_Tutorial]$
[delaneyk@amazon body1_Tutorial]$ createPatch
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : createPatch
Date   : Apr 08 2010
Time   : 15:46:37
Host   : amazon.dt.navy.mil
PID    : 21526
Case   : /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/delaneyk-1.5-dev/run/body1/boundingI
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPC
e8RASModel11printCoeffsEv
Reading createPatchDict.

Using relative tolerance 0.001 to match up faces and points

Create polyMesh for time = 0

Adding new patch hull of type wall as patch 7

Moving faces from patch bow to patch 7
Moving faces from patch midbody to patch 7
Moving faces from patch stern to patch 7

Doing topology modification to order faces.

Synchronising points.
Points changed by average:0 max:0

Removing patches with no faces in them.

Removing empty patch bow at position 0
Removing empty patch midbody at position 1
Removing empty patch stern at position 2
Removing patches.
--> FOAM Warning :
    From function forces::forces(const objectRegistry& obr, const dictionary& dict)
    in file forces/forces.C at line 78
    No fvMesh available, deactivating.
Writing repatched mesh to 1

End
```

You will notice that the second to last line states "***Writing repatched mesh to 1***". The ***createPatch*** command will write the new patched surface information into a directory whose name is the first time step output of your future run, which in this case is ***1/***.

So now you need to get rid of the old ***constant*** directory and move the new ***1/*** directory to ***constant.*** In Linux this would be accomplished by:

>> mv constant orig_constant

>> mv 1 constant

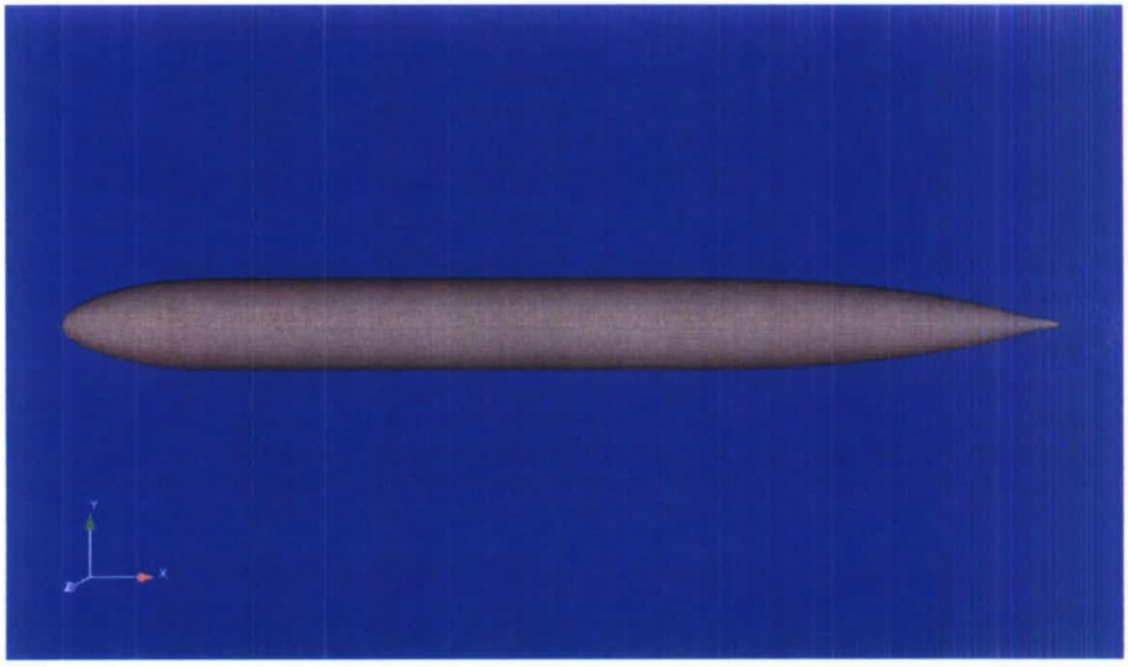
Now the ***transportProperties*** and ***RASProperties*** files need to be placed in the ***constant*** directory. The ***transportProperties*** and ***RASProperties*** files must ALWAYS be present in the ***constant*** directory when using ***simpleFoam***. In Linux this would be accomplished by:

>> mv transportProperties constant/

>> mv RASProperties constant/

Open the ***constant/polyMesh/boundary*** file and you will see that the three patches are now grouped together in the ***hull*** patch.

Now we have the geometry imported and named as we want for the run. A good next step is to export the geometry into a visual package (EnSight, ParaView, etc.) and make sure that all surfaces are grouped and labeled correctly. To export the geometry, use ***foamToEnsight*** for EnSight, ***foamToVTK*** for ParaView, and no additional command is needed for ParaFoam. So now take a minute or two and inspect your geometry in your package of choice. Your geometry should look like the Figure below, with the appropriate surface labels in the post-processor.



Material Properties

The next step is to set up the material properties for the fluid. For the simpleFoam solver, only the kinematic viscosity is required in the constant/transportProperties dictionary file. Open the transportProperties file, it should look like the screen capture below. No editing is necessary, just note that kinematic viscosity is always set in transportProperties.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "constant";
    local           "";

    class           dictionary;
    object          transportProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu                      nu [0 2 -1 0 0 0 0] 1.5151e-7;

}


// ************************************************************************* //
```

The kinematic viscosity, *nu*, is entered in as:

nu nu [0 2 -1 0 0 0] 1.5151e-7;

where *[0 2 -1 0 0 0 0]* sets the units based on [Mass Length Time Temperature Quantity Current Luminous intensity]. The kinematic viscosity dimensions are Length^2/Time or in SI units: m^2/s. The value of the kinematic viscosity is set to *1/Re* where the Reynolds number is 6.6M (*L* = *U* = 1.0). Remember that this value must always be consistent with the Reynolds number,

Re = UL/nu

0/ directory (Initial and Boundary Conditions)

Now we turn our attention to the initial and boundary conditions, which are stored in the *0/* directory.

For the ***icoFoam*** solver only ***U*** and ***p*** files are needed in the *0/* directory.

Open the ***0/U*** file; it should look like the screen capture on the following page.

In the ***0/U*** file notice:

U is a vector field quantity. All ***U*** values must be set in vector format, ***(X X X)***.

The *U* dimensions must match the variable by M,L,T,... so velocity is *[0 1 -1 0 0 0 0]*

The ***internalField*** sets the initial flow field condition for U. For this case the fluid is initially at free stream everywhere (Ux=1 and Uy,Uz = 0)

The ***boundaryField*** sets velocity boundary conditions for ALL surfaces. All surfaces must be included with proper BC's. All surface names must match the ***constant/polyMesh/boundary*** surface name exactly. The order of surfaces is not important, but the names must match identically.

The velocity boundary conditions for the five surfaces are as follows:

The ***hull*** is set with

type fixedValue;

value uniform (0 0 0);

to apply the no-slip boundary condition to the walls.

The ***farfield*** is set with

type zeroGradient;

to apply a zero velocity gradient at the farfield boundaries.

The ***inlet*** is set with

type fixedValue;

value uniform (1 0 0);

for inflow velocity of ***Ux***=1.

The ***outlet*** is set with

type zeroGradient;

to apply a zero velocity gradient at the outlet boundary.

The ***symmetry*** is set with

type symmetryPlane;

All symmetry plane boundary conditions need to have ***type symmetryPlane***.

Again, the order of the surfaces in the ***0/...*** files doesn't matter, but the names and ***types*** MUST be consistent with those listed in the ***constant/polyMesh/boundary*** file.

```
| ---------                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (1 0 0);

boundaryField
{
    hull
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    farfield
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
    symmetry
    {
        type            symmetryPlane;
    }
}


// ************************************************************************* //
```

Now open the ***0/p*** file, it should look like the screen capture below.

In the ***0/p*** file notice:

p is a scalar field quantity. All ***p*** values must be set as a scalar, ***X***, value.

The ***p*** dimensions must match the variable by M,L,T,... so pressure is [0 2 -2 0 0 0 0], because in icoFoam ***p*** is actually the pressure divided by the density, thus the SI units would be m^2/s^2.

The ***internalField*** sets the initial condition for ***p***. For this case (and most others) we do not care about the absolute value of the pressure, ***p***, so we just set it to 0 for ease.

The ***boundaryField*** sets pressure boundary conditions for ALL surfaces. All surfaces must be included with proper BC's that are consistent with the ***constant/polyMesh/boundary*** surfaces.

The pressure boundary conditions for the five surfaces are as follows:

The ***hull*** is set with

type zeroGradient;

for the no slip wall.

The ***farfield*** is set with

type zeroGradient;

to apply a zero pressure gradient at the farfield boundaries.

The ***inlet*** is set with

type zeroGradient;

to apply a zero pressure gradient at the inlet boundary.

The ***outlet*** is set with

type fixedValue;

value uniform 0.0;

to set a reference pressure boundary at the outlet.

Note: At least one boundary in all domains must have a set pressure.

The ***symmetry*** is set with

type symmetryPlane;

All symmetry plane boundary conditions need to have ***type symmetryPlane***.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    hull
    {
        type            zeroGradient;
    }
    farfield
    {
        type            zeroGradient;
    }
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0.0;
    }
    symmetry
    {
        type            symmetryPlane;
    }
}
```

Now open the ***0/nuTilda*** file, it should look like the screen capture below.

In the ***0/nuTilda*** file notice:

nuTilda is a scalar field quantity. All ***nuTilda*** values must be set as a scalar, ***X***, valuc.

The ***nuTilda*** dimensions must match the variable by M,L,T,... so viscosity is [0 2 -1 0 0 0 0], thus the SI units would be m^2/s.

The ***internalField*** sets the initial condition for ***nuTilda***. For this case it is set to 1e-8, which is ~10% of the kinematic viscosity.

The ***boundaryField*** sets ***nuTilda*** conditions for ALL surfaces. All surfaces must be included with proper BC's that arc consistent with the ***constant/polyMesh/boundary*** surfaces.

The ***nuTilda*** boundary conditions for the five surfaces are as follows:

The ***hull*** is set with

type fixedValue;

value uniform 0;

for the no slip wall.

The ***farfield*** is set with

type zeroGradient;

to apply a zero ***nuTilda*** gradient at the farfield boundaries.

The ***inlet*** is set with

type fixedValue;

value uniform 1e-8;

to set 10% of the kinematic viscosity at the inlet boundary.

The ***outlet*** is set with

type zeroGradient;

to apply a zero ***nuTilda*** gradient at the outlet boundary.

The ***symmetry*** is set with

type symmetryPlane;

All symmetry plane boundary conditions need to have ***type symmetryPlane***.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      nuTilda;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 1e-08;

boundaryField
{
    hull
    {
        type            fixedValue;
        value           uniform 0;
    }
    farfield
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 1e-08;
    }
    outlet
    {
        type            zeroGradient;
    }
    symmetry
    {
        type            symmetryPlane;
    }

}


// ************************************************************************* //
```

Now open the ***0/nut*** file, it should look like the screen capture on the next page.

This file is required by the ***simpleFoam*** solver when the Spalart-Allmaras turbulence model is used, but its turbulent viscosity input values are not used in the calculations. It is most likely a bug in the code. Nonetheless, valid ***types*** and path names are required in the ***nut*** file.

In the ***0/ nut*** file notice:

nut is a scalar field quantity. All ***nut*** values must be set as a scalar, ***X***, value.

The ***nut*** dimensions must match the variable by M,L,T,... so viscosity is [0 2 -1 0 0 0 0], thus the SI units would be m^2/s.

The ***internalField*** sets the initial condition for ***nut***. For this case it is set to 1e-6, this value is not important to the calculation, so this is simply a general ballpark value.

The ***boundaryField*** sets ***nut*** conditions for ALL surfaces. All surfaces must be included with proper BC's that are consistent with the ***constant/polyMesh/boundary*** surfaces.

The ***nut*** boundary conditions for the five surfaces are as follows:

The ***hull*** is set with

type zeroGradient;

to apply a zero ***nut*** gradient at the no slip boundary.

The ***farfield*** is set with

type zeroGradient;

to apply a zero ***nut*** gradient at the farfield boundaries.

The ***inlet*** is set with

type fixedValue;

value uniform 1e-6;

to set turbulent viscosity at the inlet boundary.

The ***outlet*** is set with

type zeroGradient;

to apply a zero ***nut*** gradient at the outlet boundary.

The ***symmetry*** is set with

type symmetryPlane;

All symmetry plane boundary conditions need to have ***type symmetryPlane***.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 1e-06;

boundaryField
{
    hull
    {
        type            zeroGradient;
    }
    farfield
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 1e-06;
    }
    outlet
    {
        type            zeroGradient;
    }
    symmetry
    {
        type            symmetryPlane;
    }
}
```

system/ directory (Solver Settings)

Now we will look at some of the solver settings and controls that are located in the ***system/*** directory. We will focus on the ***controlDict***, ***fvSolution***, ***fvSchemes***, and ***decomposeParDict*** files. We already used the ***createPatchDict*** to merge multiple surfaces.

Open the ***system/controlDict*** dictionary file. It should look like the screen capture below. The ***controlDict*** file sets all of the run-time parameters and output information. This is also

where run-time libraries and functions, such as force outputs over a patch and dynamic mesh libraries are specified.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

// FoamX Case Dictionary.

FoamFile
{
    version         2.0;
    format          ascii;

    root            "tutorial";
    case            "body1";
    instance        "system";
    local           "";

    class           dictionary;
    object          controlDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

libs ("libnavyFiniteVolume.so" "libnavyIncompressibleRASModels.so");

application     simpleFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1500;

deltaT          1;

writeControl    timeStep;

writeInterval   500;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression compressed;

timeFormat      general;

timePrecision   6;

graphFormat     raw;

runTimeModifiable yes;
```

(Screen capture continucd on ncxt page...)

```
functions
(
    forces_Hull
    {
        type forces;

        //Library to load
        functionObjectLibs ("libforces.so");

        //Name of patch to integrate forces over
        patches ( hull );

         //Reference density for fluid - can be changed later ...
        rhoInf 1.0;

        //Origin for moment calculations
        CofR (0 0 0);
    }

);

// ************************************************************************* //
```

At the top, finite volume and turbulence model libraries are dynamically loaded by ***libs("..");*** .

The solver specified in ***application*** input does not matter. The solver is specified on the command line or in a script file. Thus, this is an insignificant line for our purposes.

The solver settings are fairly obvious, and more detail is provided on page U-108 of the User's Guide (http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf). For now we will only cover a broad view of the file.

We know that ***simpleFoam*** is a steady solver. Thus the solver will artificially iterate in "time", where 1 second is an iteration. Here we see that the solver start from ***startTime = 0***, and will iterate in steps of ***deltaT = 1*** until ***endTime = 1500***. The data will be written in ASCII format in directories according to ***writeInterval***. Notice that ***runTimeModifiable*** is chosen to ***yes***, this means that we can make changes to the ***controlDict*** in the middle of a run, and they will be adjusted on the fly, as opposed to having the settings set in stone for the whole calculation.

One important note is that to start a calculation from a previous solution the ***startFrom*** entry must be switched to ***latestTime***, and desired start time information (directory and BC's) must be present in the case directory. We will delve into this further later on.

Now open the ***system/fvSolution*** dictionary file. It should look like the screen capture on the next page.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.4                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "";
    local           "";

    class           dictionary;
    object          fvSolution;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{

    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-7;
        relTol           0.01;
        minIter          1;
        maxIter          200;
    };

    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-07;
        relTol           0.0;
        minIter          1;
    };

    nuTilda PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-08;
        relTol           0.01;
        minIter          1;

    };
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell 0;
    pRefValue 0;

}
```

(Screen capture continues on the next page...)

```
relaxationFactors
{
    p               0.3;
    U               0.4;
    nuTilda         0.4;
    k               0.4;
    omega           0.4;
}


// ************************************************************************* //
```

The ***fvSolution*** file contains linear solver information as well as solver algorithm settings.

The ***solvers*** section contains linear solver settings for pressure, velocity, and turbulent viscosity. Note that for this case we are using preconditioned conjugate gradient solvers (***PCG*** for symmetric matrices and ***PBiCG*** for asymmetric matrices), but we also commonly use multi-grid solvers (GAMG, AAMG, etc.). The solver tolerance and relative tolerance settings are not important right now. The ***minIter*** command sets a minimum number of times the linear solver will iterate on a variable. It is usually recommended that the user always set a minimum number of iterations > 0 to prevent the solver from prematurely not solving for a variable (we recommend ***minIter*** = 1).

Below the ***solvers*** section are ***SIMPLE*** algorithm control settings. These ***SIMPLE*** settings are not particularly useful to the user at this time, so only a broad view of what each setting means is given. Also, note that the ***PISO*** algorithm must be used for all transient solvers and the ***SIMPLE*** algorithm must be used for all steady-state solvers. For this case we have ***nNonOrthogonalCorrectors*** set to 0, which means that we will not solve the pressure equation more than once per iteration. Note for future runs, if the pressure residuals are increasing and the solution is diverging/blowing up, ***nNonOrthogonalCorrectors*** can be increased to iterate the pressure equation more and may lead to successful solution convergence. Notice that we have set cell number 0 as our reference cell, where the reference value is 0. This is the reference pressure for the incompressible solver.

Finally, the ***relaxationFactors*** section is where under-relaxation factors for each variable are specified. Typical pressure values are 0.1-0.4 and typical velocity and turbulence quantity values are 0.4-1.0. Higher values correspond to quicker solution advancement, but will be more unstable (greater chance of solution divergence).

Now open the ***system/fvSchemes*** dictionary file. It should look like the screen capture below.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/
FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "system";
    local           "";

    class           dictionary;
    object          fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default                             steadyState;
}

gradSchemes
{
    default                             Gauss linear;

}

divSchemes
{
    default                             none;

    div(phi,U)                          Gauss linearUpwind cellLimited Gauss linear 1.0;
    div(phi,nuTilda)                    Gauss upwind;
    div((nuEff*dev(grad(U).T())))       Gauss linear;
}

laplacianSchemes
{
    default                             Gauss linear corrected;
}

interpolationSchemes
{
    default                             linear;
}

snGradSchemes
{
    default                             corrected;
}

fluxRequired
{
    default                             no;
    p;
}
```

Many of the ***fvSchemes*** settings are not particularly useful to the user at this time, so only a broad view of the settings is given here. For more detail on these settings consult page U-110 of the User's Guide.

The ***fvSchemes*** file sections declares the following settings:

ddt → time discretization

gradSchemes → gradient term discretizations

divSchemes → divergence terms discretization

laplacianSchemes → Laplacian terms discretization

interpolationSchemes → interpolation of values from cell centers to cell face centers

snGradSchemes → surface normal gradient evaluation at cell faces

fluxRequired → lists fields where flux is generated in the application

Some ***fvSchemes*** notes:

(1) Because ***simpleFoam*** is a steady solver ***ddtSchemes*** default is set to ***steadyState***.

(2) The ***div(phi,U)*** term is the convective velocity term, and "***Gauss linearUpwind cellLimited Gauss linear 1.0***" corresponds to 2nd order upwind.

(3) The ***div(phi, nuTilda)*** term is the convective turbulent viscosity term, and "***Gauss upwind***" corresponds to 1st order upwind.

(4) The ***div((nuEff*dev(grad(U).T())))*** term requires a ***gradSchemes*** input, but is placed in ***divSchemes***. This is probably a bug.

For all of the ***fvSchemes*** fields a ***default*** value can be specified and only exceptions to the default setting would need to be specified, or ***default*** can be set to ***none*** which means that the user must enter all values for the appropriate variables themselves.

Now open the ***system/decomposeParDict*** dictionary file. It should look like the screen capture on the next page.

There is a lot in the ***decomposeParDict*** that is beyond the scope of this tutorial, but the important thing to notice is that the mesh will be split into 12 partitions (***numberOfSubdomains 12;***) using the ***metis*** method.

The number of 1's in the ***metisCoeffs*** → ***processorWeights*** section must match the number in ***numberOfSubdomains***.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "";
    local           "";

    class           dictionary;
    object          decomposeParDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


numberOfSubdomains 12;

method          metis;

simpleCoeffs
{
    n               (2 2 1);
    delta           0.001;
}

hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}

metisCoeffs
{
    processorWeights
    (
        1 1 1 1 1 1 1 1 1 1 1 1
    );
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

// ************************************************************************* //
```

Next execute the settings from ***decomposeParDict*** by entering "***decomposePar***" on the command line.

Upon completion of the domain decomposition, your directory will have twelve new files (***processor0*** → ***processor11***), which all correspond to the decomposed domain. Your case directory should look like the screen capture below.

```
[delaneyk@amazon body1_Tutorial]$ l
total 237M
-rw-r--r--  1 delaneyk users 237M Apr  8 15:31 body1_Box-ASCII.fluent.cas
drwxr-xr-x  3 delaneyk users   21 Apr  8 15:49 orig_constant
drwxr-xr--  2 delaneyk users   46 Apr  8 16:41 0
drwxr-xr-x  3 delaneyk users   14 Apr  8 16:45 forces_Hull
drwxr-xr-x  3 delaneyk users   67 Apr  8 16:47 constant
drwxr-xr-x  2 delaneyk users  102 Apr  8 16:50 system
-rw-r--r--  1 delaneyk users  650 Apr  8 16:51 oFOAM.scp
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:52 processor0
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:52 processor1
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor2
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor3
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor4
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor5
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor6
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor7
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor8
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor9
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor10
drwxr-xr-x  4 delaneyk users   29 Apr  8 16:53 processor11
[delaneyk@amazon body1_Tutorial]$
```

Running the Case

We are now ready to run our case. To execute this case on a cluster a script file is needed. For example purposes the script file ***oFOAM.scp*** is shown below.

Notice that the 12 partition mesh will be run on 3 nodes with 4 processors per node. The application ***simpleFoam*** is also specified in this file.

-It is now time to run the job, so in this case we type:

>>qsub oFOAM.scp

into the command line. A file named ***log*** will contain all of the run information that would normally be output in a screen dump.

Remember that at the bottom of our ***controlDict*** file, we specified a function named ***forces_Hull*** of ***type forces***. This file calculates forces over the patch specified by ***patches*** (***hull*** in our case), and places them in a directory named ***forces_Hull*** under a time file name that corresponds to ***startTime***.

Now let the file run out until its ***endTime*** of 1500 iterations.

```
#PBS -j oe
#PBS -o ./amazon.out
#PBS -e ./amazon.err
#PBS -S /bin/csh
#PBS -N SA_b1
#PBS -l nodes=3:ppn=4
#PBS -l walltime=42:00:00
#PBS -V
echo "cd to the directory"
cd $PBS_O_WORKDIR

setenv OPENFOAM_NP 12

echo "define parameters in exec statement"

set APPLICATION="simpleFoam"
set ROOT="."
set CASE="Body_1_Tutorial"

echo "The current shell is $SHELL"
echo "Number of processors: $OPENFOAM_NP"
echo "Executing           : $APPLICATION $ROOT $CASE"
echo "Working directory   : $PBS_O_WORKDIR"
echo "The shell limits are:"
limit
echo "Starting executable...."


mpirun -machinefile $PBS_NODEFILE -np $OPENFOAM_NP $APPLICATION -parallel > ./log
```

After the case has completed, by running 1500 iterations open up the ***forces_Hull/1*** file. Let's just say for tutorial purposes that the forces have not converged to our satisfaction, and we want to run the case out further for an additional 2500 iterations.

To restart the case make the following changes in the ***system/controlDict*** file:

(1) Change: ***startFrom startTime;*** → ***startFrom latestTime;***

(2) Change: ***endTime 1500;*** → ***endTime 4000;***

-Now restart the calculation with

>>qsub oFOAM.scp

Notice that the ***log*** file will be written over (so make a copy in the future if you wish to keep the original ***log*** file). Also notice that forces are now being output under ***forces/1501*** file, and the original forces are still kept under ***forces/1***.

Let the case run out to completion after 4000 iterations.

Now open the ***log*** file. Some observations:

You can see that the solver started from time equal to 1500 and iterated until 4000.

For each iteration the momentum equation (***Ux***, ***Uy***, and ***Uz***) is solved first, then the continuity equation (***p***), and finally the turbulent quantity (***nuTilda***).

For each variable linear solver we can see the initial residual, final residual, and the number of iterations it took to drop from the initial to the final residual. We set all of these tolerances and iteration criteria in the ***system/fvSolution*** dictionary file.

There are also continuity error reports.

The best way to typically monitor the solution is to make sure that the velocity magnitude stays at a reasonable number, and make sure that initial pressure residuals are decreasing or are holding steady at an acceptable value.

The last line of the time iteration produces execution and clock time information. This is useful in gauging the efficiency of your solution.

Post-Processing

Notice that there are many time directories in your processor directories. Each of these directories contains output information for their respective time step.

To reconstruct the data from the decomposed processors use the command

>> reconstructPar –latestTime

The *–latestTime* means only reconstruct the last time in the ***processor**** files. The command *–time time#* will reconstruct for a specific time (*time#*) only. If only *reconstructPar* is specified, then all time directories in the ***processor**** files will be reconstructed.

To look at the post-processed results simply type the following commands, depending on the post-processing tool of choice:

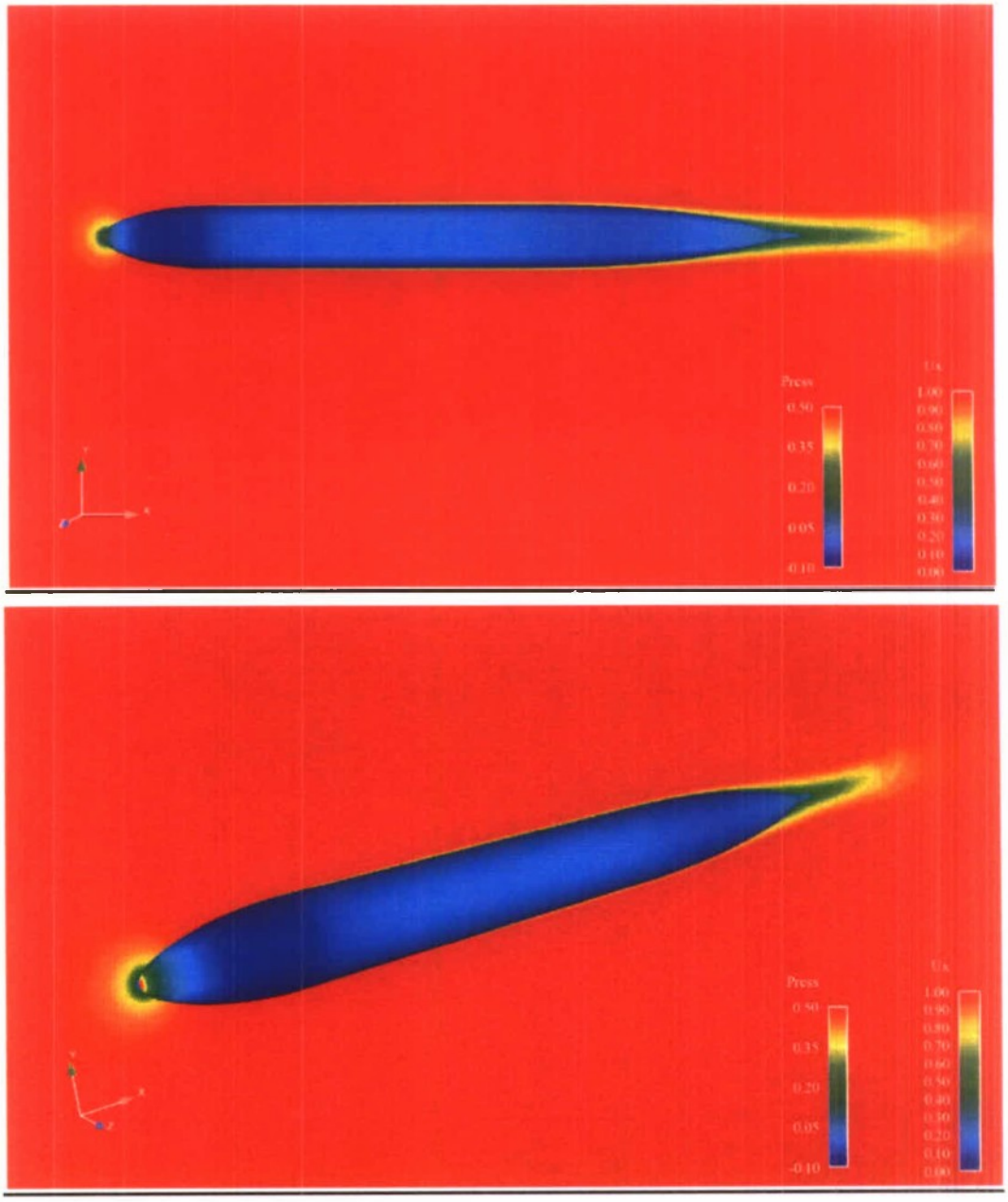
>> foamToEnSight -latestTime → to look at the results in EnSight

>> foamToVTK -latestTime → to look at the results in ParaView

where the command *–latestTime* is used to only look at the results from the last output time step. To look at the results for all time steps simply leave off the *–latestTime* command, and to look at the results for a specific time (i.e. 0.005) use *–time 0.005*.

To look at the results in ParaFoam, no additional commands are needed, simply open ParaFoam in the case directory.

Your results should look like the axial velocity (*Ux*) and pressure (*Press*) contours below.


Press
0.50
0.35
0.20
0.05
-0.10
Ux
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00

Appendix E: ransFSNavyFoam Wigley Hull Tutorial

This tutorial involves using the turbulent, transient, incompressible, multi-phase solver for the Wigley hull. Although this is a transient solver, this case will NOT be run time accurate. Only half the body is solved, as symmetry is assumed. First, we will go over pre-processing and case setup, then we will run the test case, and finally we will look at some post-processed results.

For more detailed information on the OpenFOAM code and settings consult the User's Guide: http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf.

Pre-Processing and Case Setup

Your initial directory should look like the screen capture below.

```
total 4.0K
drwxr-xr-x  3 delaneyk users 128 Jul 13 14:50 constant
drwxr-xr-x  2 delaneyk users 100 Jul 13 14:50 system
drwxr-xr-x  2 delaneyk users  72 Jul 13 14:50 0
-rw-r--r--  1 delaneyk users 646 Jul 13 14:50 oFOAM.scp
```

constant/ directory

In the previous tutorials the mesh needed to be imported into OpenFOAM from a 3rd party mesh generator. However, for this case the mesh has already been imported, so you will notice the ***polymesh/*** folder is already present in the ***constant/*** directory. Open up your ***constant/polyMesh/boundary*** file, it should look like the following screen capture. Notice that the ***hull*** surface is of type ***wall*** (viscous surfaces must always be of type ***wall***), the ***centerplane*** is of type ***symmetryPlane*** (symmetry planes must always be of type ***symmetryPlane***), and the rest of the surfaces are of type ***patch***.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5.x                                 |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
7
(
    hull
    {
        type            wall;
        nFaces          3074;
        startFace       762555;
    }
    centerplane
    {
        type            symmetryPlane;
        nFaces          5858;
        startFace       765629;
    }
    bottom
    {
        type            patch;
        nFaces          3364;
        startFace       771487;
    }
    farfield
    {
        type            patch;
        nFaces          8932;
        startFace       774851;
    }
    top
    {
        type            patch;
        nFaces          3364;
        startFace       783783;
    }
    inlet
    {
        type            patch;
        nFaces          2233;
        startFace       787147;
    }
    outlet
    {
        type            patch;
        nFaces          2233;
        startFace       789380;
    }
)
// ************************************************************************* //
```

Now run the ***checkMesh*** command for two reasons:

1. to make sure the mesh was imported correctly

2. to asses the quality of the mesh for the OpenFOAM solver

Your ***checkMesh*** output should look like the screen captures on the next pages.

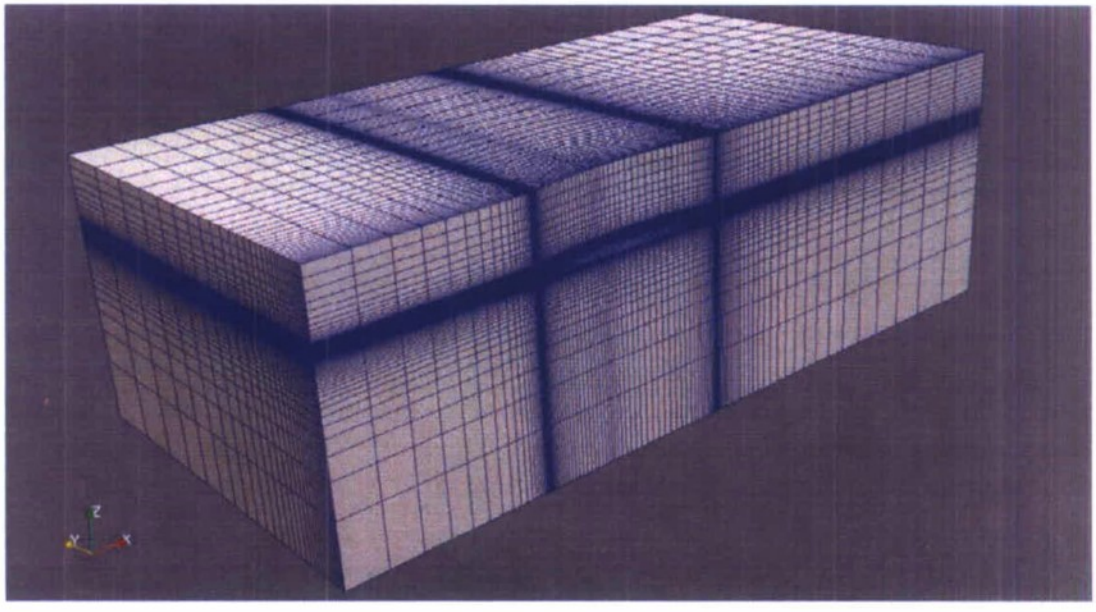
```
[delaneyk@amazon wigley_tutorial]$ checkMesh
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : checkMesh
Date   : Jun 21 2010
Time   : 10:56:06
Host   : amazon.dt.navy.mil
PID    : 29416
Case   : /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/delaneyk-1.5-dev/run/wigley/tutorial/wigley_tutorial
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPOpt/libnavyFiniteV
olume.so: undefined symbol: _ZN4Foam6upwindIdE8typeNameE
--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPOpt/libnavyIncompr
essibleRASModels.so: undefined symbol: _ZN4Foam14incompressible8RASModel11printCoeffsEv
Create polyMesh for time = constant

Time = constant

Mesh stats
    points:           273780
    faces:            791613
    internal faces:   762555
    cells:            259028
    boundary patches: 7
    point zones:      0
    face zones:       0
    cell zones:       0

Number of cells of each type:
    hexahedra:     259028
    prisms:        0
    wedges:        0
    pyramids:      0
    tet wedges:    0
    tetrahedra:    0
    polyhedra:     0

Checking topology...
    Boundary definition OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).

Checking patch topology for multiply connected surfaces ...
    Patch             Faces    Points   Surface topology
    hull              3074     3186     ok (non-closed singly connected)
```

```
    centerplane          5858     6105     ok (non-closed singly connected)
    bottom               3364     3510     ok (non-closed singly connected)
    farfield             8932     9126     ok (non-closed singly connected)
    top                  3364     3510     ok (non-closed singly connected)
    inlet                2233     2340     ok (non-closed singly connected)
    outlet               2233     2340     ok (non-closed singly connected)

Checking geometry...
    This is a 3-D mesh
    Overall domain bounding box (-4 -9.40395e-38 -4) (12 8 1.2)
    Mesh (non-empty) directions (1 1 1)
    Mesh (non-empty, non-wedge) dimensions 3
    Boundary openness (-2.25653e-16 -6.20107e-16 -3.37491e-16) Threshold = 1e-06 OK.
    Max cell openness = 3.23887e-16 OK.
    Max aspect ratio = 750.787 OK.
    Minumum face area = 7.99769e-06. Maximum face area = 1.44004.  Face area magnitudes OK.
    Min volume = 3.21742e-08. Max volume = 1.44.  Total volume = 664.872.  Cell volumes OK.
    Mesh non-orthogonality Max: 78.8601 average: 12.7201 Threshold = 70
   *Number of severely non-orthogonal faces: 18.
    Non-orthogonality check OK.
  <<Writing 18 non-orthogonal faces to set nonOrthoFaces
    Face pyramids OK.
    Max skewness = 1.46173 OK.

Mesh OK.

End
```

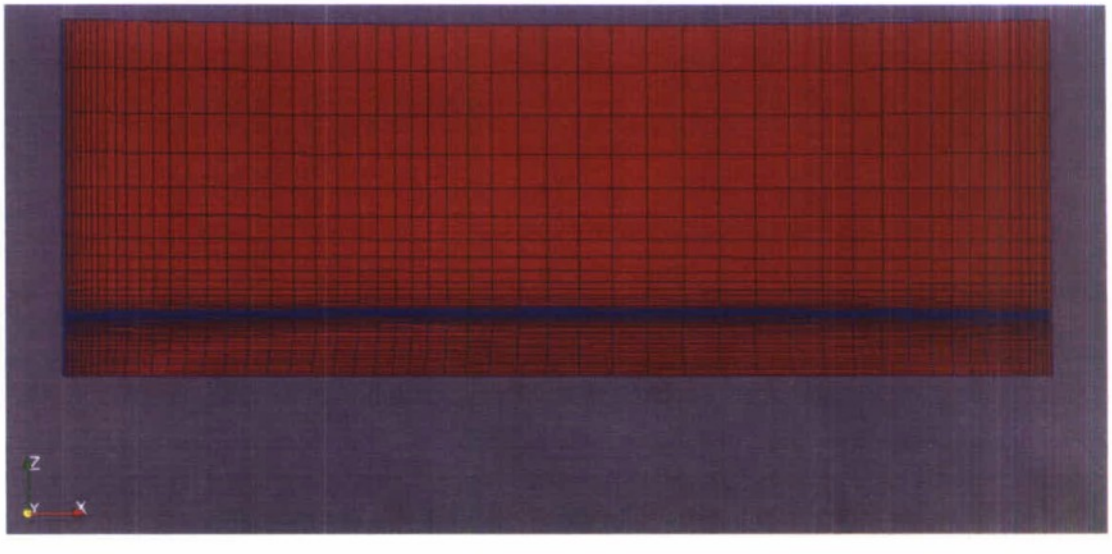
You will notice that there are 18 "severely non-orthogonal faces." As has been mentioned in previous tutorials, OpenFOAM's ***checkMesh*** is very harsh. Sometimes it is not possible to create a mesh without any high aspect ratio, non-orthogonal, or skewed cells. In fact, most meshes created will contain bad cells, and run fine. However, at some point (which is not quantitatively clear) the mesh will be so poor it either won't run, or it will take a long time to run. There aren't exact guidelines on OpenFOAM mesh quality; it simply takes experience running various meshes.

Another good initial step is to export the geometry into a visual package (EnSight, ParaView, etc.) and make sure that all surfaces are grouped and labeled correctly. To export the geometry, use ***foamToEnsight*** for EnSight, ***foamToVTK*** for ParaView, and no additional command is needed for ParaFoam. So now take a minute or two and inspect your geometry in your package of choice. Your geometry should look like the pictures on the next page, with the appropriate surface labels. This mesh is meant for instructional purposes only as you will notice that the mesh is very coarse.

Entire Domain:



Side View of Hull:



The constant/RASProperties file is the same as in the previous tutorials and will not be covered here.

Open the constant/transportProperties file, it should look like the screen capture below. No editing is necessary. However, notice that the multi-phase solver requires density (rho) and

kinematic viseosity (nu) for both the water (phase 1) and the air (phase 2). Additionally the surface tension (sigma) is input at the bottom. The surfaee tension could probably be negleeted for this case of a surface ship (sigma = 0.0), but it must always be included at the bottom of the transportProperties file.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "wigley";
    instance        "";
    local           "";

    class           dictionary;
    object          transportProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

phase1
{
    transportModel  Newtonian;
    rho             rho [1 -3 0 0 0 0 0] 1000;
    nu              nu [0 2 -1 0 0 0 0]  1e-06;
}

phase2
{
    transportModel  Newtonian;
    rho             rho [1 -3 0 0 0 0 0] 1;
    nu              nu [0 2 -1 0 0 0 0]  1.48e-05;
}

sigma           sigma [1 0 -2 0 0 0 0] 0.07;


// ************************************************************************* //
```

The multi-phase solver also requires a constant/environmentalProperties file, whieh has not been required in the previous tutorials. This file eontains information on the gravity veetor as can be seen on the next screen capture. An important note for free surfaee flow is to make sure that the gravity and velocity (from the 0/U file) coincide with the desired Froude number aeeording to the definition:

$$Fr = \frac{U}{\sqrt{gL}}$$

For this tutorial the Wigley Hull will be run at a Froude number of 0.289, eorresponding to a Reynolds number of 905,000.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "wigley";
    instance        "";
    local           "";

    class           dictionary;
    object          environmentalProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

g               g [0 1 -2 0 0 0 0] (0 0 -9.81);


// ************************************************************************* //
```

0/ directory(Initial and Boundary Conditions)

Now we turn our attention to the initial and boundary conditions, which are stored in the *0/* directory. Again, many of the basic concepts stored in the *0/* directory have been covered in previous tutorials, thus only new concepts will be covered here. However, it is worth repeating that ALL surface names in the *0/...* files must match the names from the ***constant/polyMesh/boundary*** file.

For the ***ransInterNavyFoam*** solver with the SST k-omega turbulence model only ***U, k, omega, pd,*** and ***gamma*** files are needed in the *0/* directory.

Open the *0/U* file; it should look like the screen capture on the next page. The ***hull*** is set to a no-slip boundary condition, the ***inlet, farfield,*** and ***bottom*** boundaries are set to slip boundary conditions to simulate the coordinate system fixed to the hull as would be the case in tow tank tests.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0.905 0 0);

boundaryField
{
    hull
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    centerplane
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            fixedValue;
        value           uniform (0.905 0 0);
    }
    farfield
    {
        type            fixedValue;
        value           uniform (0.905 0 0);
    }
    top
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform (0.905 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
```

Both the 0/k and 0/omega files are set up similar to the previous tutorials, and should look like the following screen captures.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      omega;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 -1 0 0 0 0];

internalField   uniform 400;

boundaryField
{
    hull
    {
        type            zeroGradient;
    }
    centerplane
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            fixedValue;
        value           uniform 400;
    }
    farfield
    {
        type            fixedValue;
        value           uniform 400;
    }
    top
    {
        type            fixedValue;
        value           uniform 400;
    }
    inlet
    {
        type            fixedValue;
        value           uniform 400;
    }
    outlet
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
```

The multi-phase solver requires a ***0/gamma*** file which represents the volume fraction (***gamma*** = 0 = air and ***gamma*** = 1 = water). The solver is using the Volume of Fluid (VOF) method to solve for both the air and water.

Notice that all of the bottom and outlet boundaries are set to zero gradient. The top is set to ***inletOutlet***, which switches between a fixed value and zero gradient condition depending on the direction of flow across the boundary. The ***inlet*** and ***farfield*** are set to the ***calmWater*** boundary condition, which keeps the air-water interface at a constant height along the boundary. The ***calmWater*** condition is especially important in avoiding artificial waves at the inlet and side of the domain during mesh motion calculations. The ***centerplane*** is set to ***symmetryPlane***.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      gamma;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    hull
    {
        type            zeroGradient;
    }

    centerplane
    {
        type            symmetryPlane;
    }

    bottom
    {
        type            zeroGradient;
    }

    farfield
    {
        type            calmWater;
        valueAbove      0;
        valueBelow      1;
        elevation       0;
        axis            z;
        value           uniform 0;
    }

    top
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
```

(screen output continues on the next page...)

```
    inlet
    {
        type            calmWater;
        valueAbove      0;
        valueBelow      1;
        elevation       0;
        axis            z;
        value           uniform 0;
    }

    outlet
    {
        type            zeroGradient;
    }
}


// ************************************************************************* //
```

Now open the ***0/pd*** file and notice that for the multi-phase solver pressure is in terms of the variable ***pd***, as opposed to ***p*** for the single phase solver. For the single phase solver the pressure (***p***) is a relative pressure, whereas a more precise (***pd***) pressure is solved for in ***ransFSFoam***. The ***top*** and ***outlet*** have the pressure set to 0 and the rest are at zero gradient and symmetry plane.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      pd;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    hull
    {
        type            zeroGradient;
    }
    centerplane
    {
        type            symmetryPlane;
    }
    bottom
    {
        type            zeroGradient;
    }
    farfield
    {
        type            zeroGradient;
    }
    top
    {
        type            fixedValue;
        value           uniform 0;
    }
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
}


// ************************************************************************* //
```

System/ Folder (Solver Settings)

Now we will look at some of the solver settings and controls that are located in the ***system/*** directory, which contains: ***controlDict***, ***setFieldsDict***, ***fvSolution***, ***fvSchemes***, and ***decomposeParDict*** files.

The ***system/decomposeParDict*** dictionary file was covered extensively in the ***simpleFoam*** tutorial, thus will not be discussed here.

Open the ***system/controlDict*** dictionary file. It should look like the screen capture below.

The solver settings are fairly obvious, and more detail is provided on page U-108 of the User's Guide (http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf). For now we will only cover a broad view of the file.

The solver specified in ***application*** input does not matter. The solver is specified on the command line or in a script file. Thus, this is an insignificant line for our purposes.

The ***ransFSFoam*** solver is a transient solver, thus it requires ***maxCo*** and ***maxDeltaT*** inputs that specify the maximum possible Courant (CFL) number and time step, respectively. When the ***adjustableTimeStep*** is set to ***yes*** the time step specified by ***deltaT*** is ignored and the time step size is chosen by the maximum Courant number set by ***maxCo***.

The ***maxCo*** (CFL) command is very important to solution stability. There is no single value that is used for all cases, and in most cases the user will start out with a low CFL number and then ramp it up once the initial solution transients die out. This is a parameter that the user will have to gain experience over time to learn the best solution strategy. For now we will start with CFL = 5.0 and leave it as such throughout the solution, but it is not unusual to start cases out with CFL as low as 1 and ramp it up into the 100's. The ***controlDict*** is read continuously during the calculation, thus the CFL can be changed on the run without having to stop the run.

Near the bottom, finite volume and turbulence model libraries are dynamically loaded by ***libs("..");*** .

At the bottom the ***hullForce*** library is loaded, which calculates forces and moment over the hull surface. The moments are taken about the point specified by ***COR***.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.4                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "wigley";
    instance        "";
    local           "";

    class           dictionary;
    object          controlDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application             interFoam;

startFrom               startTime;

startTime               0;

stopAt                  endTime;

endTime                 50;

deltaT                  0.01;

writeControl            runTime;

writeInterval           10;

purgeWrite              0;

writeFormat             ascii;

writePrecision          6;

writeCompression        uncompressed;

timeFormat              general;

timePrecision           6;

runTimeModifiable       yes;

adjustTimeStep          yes;

maxCo                   5.0;

libs ("libnavyFiniteVolume.so" "libnavyIncompressibleRASModels.so" "libmyDynamicFvMesh.so");

maxDeltaT               5.e-2;
```

(screen output continues on the next page...)

```
functions
(
    hullForce
    {
        type        hullForce;

        // Where to load it from (if not already in solver)
        functionObjectLibs ("libhullForce.so");

        patches     ( hull );

        CofR        (0.5 0 0);
    }
);


// ************************************************************************* //
```

Now open the ***system/fvSolution*** dictionary file. It should look like the screen capture on the next pages.

The ***fvSolution*** file contains linear solver information as well as solver algorithm settings.

Notice that we are using multi-grid (GAMG) linear solvers for all of the pressure terms instead of the conjugate gradient (PCG) solvers from the previous tutorials. The linear solver settings and criteria are explained in further detail in the User's Guide. The important part of the solvers is noticing the tolerance and relative tolerance (***relTol***) which determine when the solver will stop iterating.

pCorr is an initial pressure calculation that is done before the first iteration only for this case (as can be seen later on in the ***log*** file). If the mesh were moving there would be a ***pCorr*** loop for each iteration.

The transient solver requires the ***PISO*** algorithm as opposed to the ***SIMPLE*** algorithm that is required for steady solvers. Most of the PISO settings (correctors) specify the number of iterations and subiterations for parameters like velocity, pressure, and gamma. Outer correctors loop through all linear solvers (***U***, ***k***, ***omega***, ***pd***, and ***gamma***), non-orthogonal correctors loop through the pressure equation, and gamma correctors and subcycles loop through the volume fraction.

For future purposes, the user is encouraged to change the various PISO settings and look at the ***log*** file to see how these settings effect the solution iterations. Solutions on high quality meshes will require less correctors and subcycles, while for poor meshes it may be necessary to have more correctors to achieve a solution. The higher the number of correctors and cycles the solution will be more stable; however, iteration time will increase rapidly. There is no "correct" answer for each ***PISO*** parameter.

The ***cGamma*** parameter specifies the sharpness of the interface (0 = less sharp and 1 = most sharp). ***CoGamma*** refers to the gamma solution advancement. For now the user should simply leave ***cGamma*** and ***CoGamma*** at 0 and 0.5 for all cases.

For the time being, make sure that your settings look like the screen. More detail on the ***PISO*** settings is provided on page U-117 of the User's Guide (http://foam.sourceforge.net/doc/Guides-a4/UserGuide.pdf).

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

// FoamX Case Dictionary.

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "wigley";
    instance        "";
    local           "";

    class           dictionary;
    object          fvSolution;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr GAMG
    {
        tolerance        1e-4;
        relTol           0;
        minIter          1;
        maxIter          25;

        smoother         DICGaussSeidel;
        nPreSweeps       0;
        nPostSweeps      2;
        nBottomSweeps    2;

        cacheAgglomeration false;
        nCellsInCoarsestLevel 10;
        agglomerator     faceAreaPair;
        mergeLevels      1;
    };


    pd GAMG
    {
        tolerance        1e-7;
        relTol           0.01;
        minIter          1;
        maxIter          25;

        smoother         DICGaussSeidel;
        nPreSweeps       2;
        nPostSweeps      2;
        nFinestSweeps    2;

        cacheAgglomeration false;
        nCellsInCoarsestLevel 10;
        agglomerator     faceAreaPair;
        mergeLevels      1;
    };
```

(screen output continues on the next page...)

```
    pdFinal GAMG
    {
        tolerance        1e-7;
        relTol           0.01;
        minIter          1;
        maxIter          100;

        nVcycles         2;

        smoother         DICGaussSeidel;
        nPreSweeps       2;
        nPostSweeps      2;
        nFinestSweeps    2;

        cacheAgglomeration false;
        nCellsInCoarsestLevel 10;
        agglomerator     faceAreaPair;
        mergeLevels      1;

    };


    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-09;
        relTol           0.001;
        minIter          1;
    };
    gamma PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-08;
        relTol           0;
        minIter          1;
    };
    k PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-07;
        relTol           0.01;
        minIter          1;
    };
    omega PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-07;
        relTol           0.01;
        minIter          1;
    };
}

PISO
{
    momentumPredictor           yes;
    nOuterCorrectors            2;
    nCorrectors                 1;
    nNonOrthogonalCorrectors    0;
```

```
    nGammaCorr              2;
    nGammaSubCycles         1;
    cGamma                  0;
    CoGamma                 0.5;
}

relaxationFactors
{
    pd          0.2;
    U           0.7;
    k           0.5;
    omega       0.5;
    gamma       0.5;
}

// ************************************************************************* //
```

Now open the system/fvSchemes dictionary file. The file is the same as tutorials cases except for additional divergence and flux terms. Under the divSchemes section gamma (phi and phirb) divergence terms are needed for the VOF solution. Also, pd, gamma, and pcorr flux terms are required under the fluxRequired section. Your file should look like the screen capture on the next page.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "wigley";
    instance        "";
    local           "";

    class           dictionary;
    object          fvSchemes;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default                     Gauss linear;
}

divSchemes
{
    div(rho*phi,U)              Gauss linearUpwind cellLimited  Gauss linear 1.0;

    div(phi,gamma)              Gauss vanLeer01;
    div(phirb,gamma)            Gauss interfaceCompression;

    div(phi,k)                  Gauss upwind;
    div(phi,omega)              Gauss upwind;
}

laplacianSchemes
{
    default                     Gauss linear corrected;
}

interpolationSchemes
{
    default                     linear;
}

snGradSchemes
{
    default                     corrected;
}
```

```
fluxRequired
{
    default         no;
    pd;
    pcorr;
    gamma;
}


// ************************************************************************* //
```

Now open the system/setFieldsDict dictionary file. This dictionary can be used to initially set flow field parameters over the entire domain. For now we will use it to set the domain volume fraction up appropriately. The field is initially set to air (defaultFieldValues setting gamma 0). The regions section uses boxToCell to set every cell within a box defined by the minimum and maximum rectangular points (which can extend outside the domain) to water (gamma 1).

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "system";
    local           "";

    class           dictionary;
    object          setFieldsDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue gamma 0
);

regions
(
    boxToCell
    {

        box (-10 0 -20) (20 20 0);

        fieldValues
        (
            volScalarFieldValue gamma 1
        );
    }
);


// ************************************************************************* //
```

The ***setFields*** command will alter the ***0/gamma*** file, so it is wise to make a copy of your original file. This can be done in Linux by:

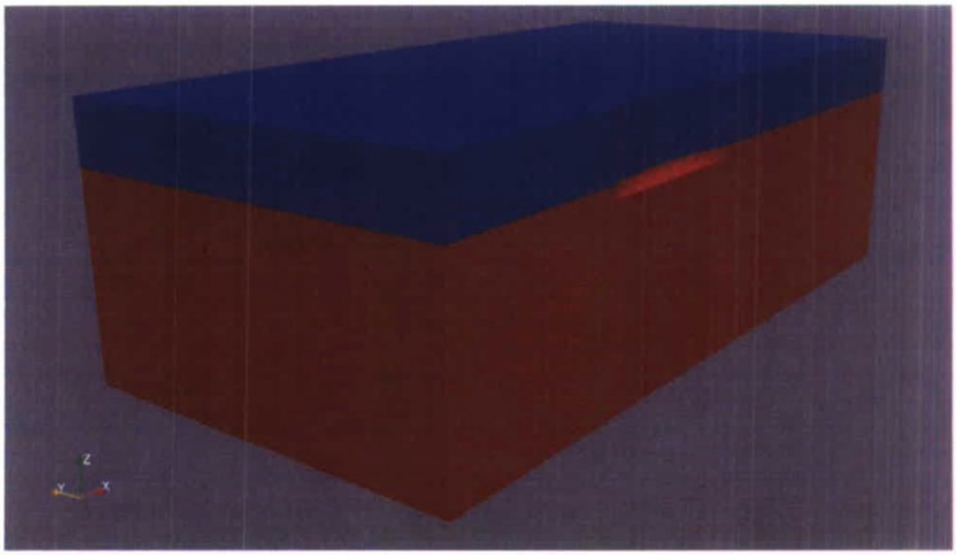
>> cp –r 0/gamma 0/Original_gamma

Now we are ready to set the initial flow field with the ***setFields*** command which executes the ***setFieldsDict*** dictionary. Enter ***setFields*** into the command line. Your output should look like the screen capture below.

```
[delaneyk@amazon wigley_tutorial]$ setFields
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: exported                              |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
Exec   : setFields
Date   : Jun 21 2010
Time   : 11:05:12
Host   : amazon.dt.navy.mil
PID    : 31380
Case   : /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/delaneyk-1.5-dev/run/wigley/tutorial/wigley_tutorial
nProcs : 1

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function dlLibraryTable::open(const fileName& functionLibName)
    in file db/dlLibraryTable/dlLibraryTable.C at line 86
    could not load /san/home/delaneyk/NavyFOAM-1.5-dev-rev995/NavyFOAM/lib/linux64GccDPOpt/libnavyIncompr
essibleRASModels.so: undefined symbol: _ZN4Foam14incompressible8RASModel11printCoeffsEv
Create mesh for time = 0

Reading setFieldsDict

Setting field default values
    Setting volScalarField gamma

Setting field region values
    Adding cells with center within box (-10 0 -20) (20 20 0)
    Setting volScalarField gamma

End
[delaneyk@amazon wigley_tutorial]$
```

It is wise to make sure that the ***setFields*** command did what it was supposed to do by viewing the results. Previous tutorials go over how to get OpenFOAM results into EnSight (***foamToEnSight***) and ParaView (***foamToVTK***).

Once the case is imported into your post-processor of choice, your domain should look like the picture below. You should have gamma equal to zero above z=0 and gamma equal to 1 below z=0.



Running the Case

Now the problem is set up correctly and ready to run. The final step is to decompose the domain by the ***decomposePar*** command.

You should now have 8 processor files (***processor0*** → ***processor7***) located in your case directory.

The final step is to submit your script (***oFoam.scp*** in this example), and then the job will run.

>>qsub oFOAM.scp

Remember that at the bottom of our ***controlDict*** file, we specified a function named ***hullForces***. This file calculates forces over the patch specified by ***patches*** (***hull*** in our case), and places them in a directory named ***hullForce*** under a time file name that corresponds to ***startTime***.

Now let the file run out until its ***endTime*** of 50 seconds.

You can open or tail the ***log*** file to monitor your solution residuals and look at your linear solution strategy that was set under the ***PISO*** section in ***system/fvSolution***. Monitoring the pressure residuals and the velocity magnitude value from iteration to iteration will give you a good idea of your solution strategy. Rapidly increasing pressure residuals or velocity magnitudes over consecutive iterations usually mean the solution is diverging.

Post-Processing

Notice that there are many time directories in your processor directories. Each of these directories contains output information for their respective time step.

To reconstruct the data from the decomposed processors use the command

>> reconstructPar –latestTime

The *–latestTime* means only reconstruct the last time in the ***processor**** files. The command *–time time#* will reconstruct for a specific time (*time#*) only. If only *reconstructPar* is specified, then all time directories in the ***processor**** files will be reconstructed.

To look at the post-processed results simply type the following commands, depending on the post-processing tool of choice:

>> foamToEnSight -latestTime → to look at the results in EnSight

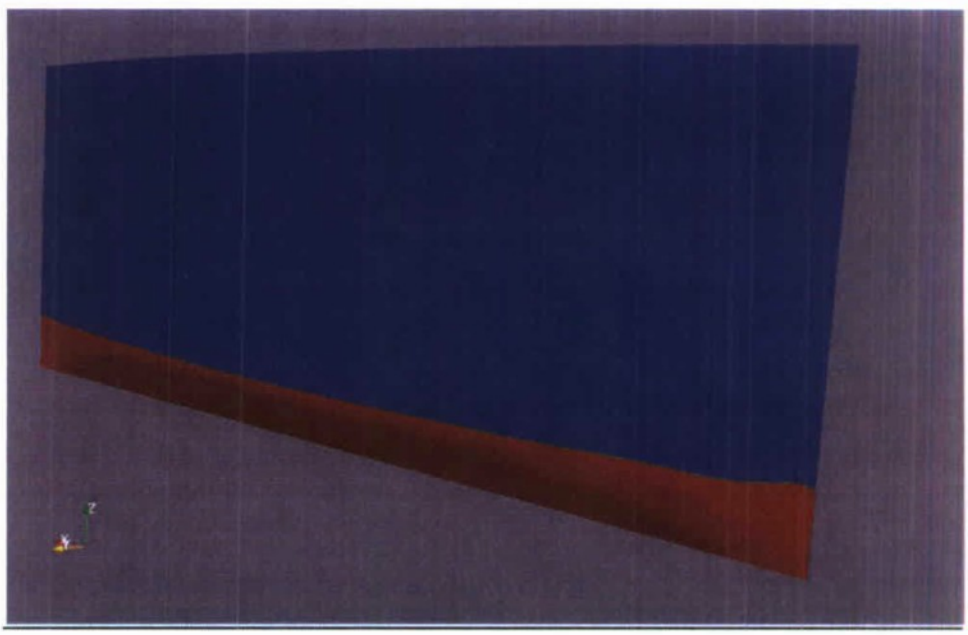
>> foamToVTK -latestTime → to look at the results in ParaView

where the command *–latestTime* is used to only look at the results from the last output time step. To look at the results for all time steps simply leave off the *–latestTime* command, and to look at the results for a specific time (ie 0.005) use *–time 0.005*.

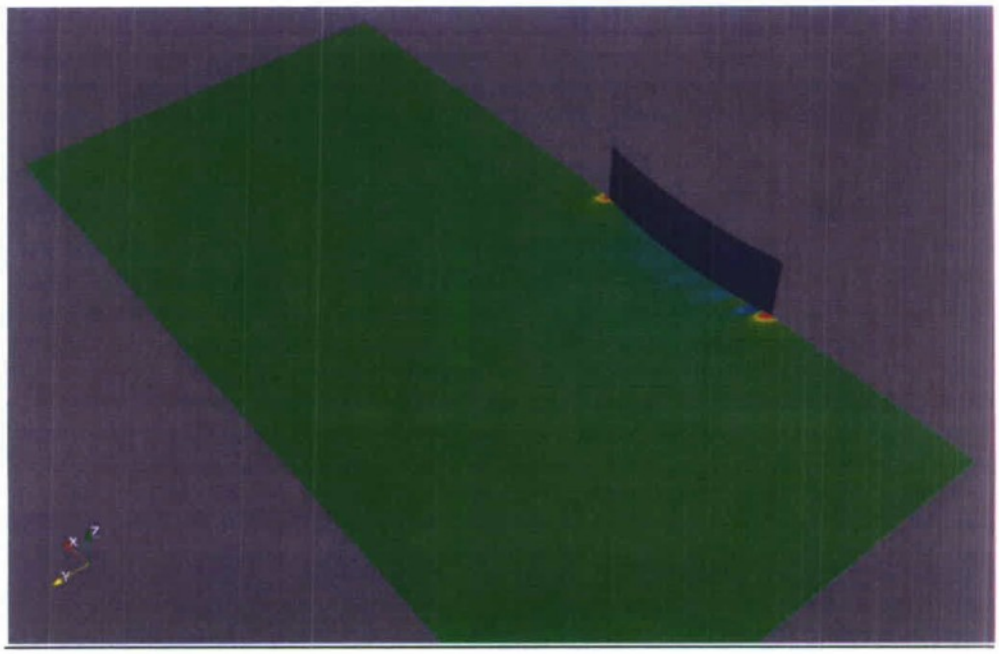
To look at the results in ParaFoam, no additional commands are needed, simply open ParaFoam in the case directory.

Your results should look like the following figures.

Bow Wave:



Free Surface Contour Plot:



THIS PAGE INTENTIONALLY LEFT BLANK

References

1. Weller, H. G., Tabor, G., Jasak, H., and Fureby, C., "A Tensorial Approach to Computational Continuum Mechanics Using Object-Oriented Techniques," *Computers in Physics*, 12(6), pp. 620 – 631, 1998.
2. Kim, S.-E., Schroeder, S. and Jasak, H., (2010), "A Multiphase CFD Framework for Predicting Performance of Marine Propulsors," *Thirteenth international Symposium on Transport Phenomena and Dynamics of Rotating Machinery*, Honolulu, Hawaii, April 4 - 9, 2010.
3. Wilcox, D.C., *Turbulence Modeling for CFD*, DCW Industries, 1998.
4. Gaskell, P.H. and Lau, A.K.C.,"Curvature-Compensated Convective Transport: SMART, a New Boundedness-Preserving Transport Algorithm," *Int. J. Numer. Methods Fluids*, Vol. 8, p. 617, 1988.
5. Leonard, B.P., "Simple High-Accuracy Resolution Program for Convective Modeling of Discontinuities," *Int. J. Numer. Meth. Fluids*, Vol. 8, pp. 1291–1318, 1988.
6. Godunov, S.K., "Finite Difference Method for Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics," *Mat. Shornik,* Vol. 47, pp. 271-306, 1959.
7. van Leer, B., "Towards the Ultimate Conservative Difference Scheme. V. A Second-Order Sequel to Godunov's Method," *J. Comput. Phys.*, Vol. 32, p. 101, 1979.
8. Jasak, H. Weller, H.G., and Gosman, A.D., "High Resolution NVD Differencing Scheme for Arbitrarily Unstructured Meshes," *Int. J. Numer. Methods Fluids*, Vol. 31, p. 431, 1999.
9. Leonard, B.P., "The ULTIMATE Conservative Difference Scheme Applied to Unsteady One-Dimensional Advection," *Comp. Meth. Appl. Mech. Eng.*, Vol. 88, pp. 17–74, 1991.
10. Ubbink O. and Issa, R.I., "Method for Capturing Sharp Fluid Interfaces on Arbitrary Meshes," *J. Comput. Phys.*, Vol. 153, pp. 26-50, 1999.
11. Muzaferija S. and Peric M., "Computation of Free Surface Flows Using Interface-Tracking and Interface-Capturing Methods," *Computational Mechanics Publications*, WIT Press, Southhampton, nonlinear water wave interaction edition, Vol. 3. pp. 59–100, 1998.
12. Park, I.R., Kim, K.S., Kim, J., and Van, S.H., "A Volume-Of-Fluid Method for Incompressible Free Surface Flows," *Int. J. Numer. Meth. Fluids*, Vol. 61, pp. 1331-1362, 2009.
13. Jasak H, and Weller H., "Interface Tracking Capabilities of the Inter-Gamma Differencing Scheme". Internal Report, Mechanical Engineering Department, Imperial College of Science, London, 1995.
14. Khosla, P.K. and Rubin, S.G., "A Diagonally Dominant Second-Order Accurate Implicit Scheme," *Comput. Fluids*, Vol. 2, p. 207, 1974.
15. Hayase, T., Humphrey, J.A.C., and Greif, R., "A Consistently Formulated QUICK Scheme for Fast and Stable Convergence Using Finite-Volume Iterative Calculation Procedures," *J. Comput. Phys.* Vol. 98, p. 108, 1992.
16. Holmes, D.G. and Connell, S.D., "Solution of the 2D Navier-Stokes Equations on Unstructured Adaptive Grids," AIAA-1989-1932, 1989.
17. Frink, N.T., "Recent Progress Toward a Three-Dimensional Unstructured Navier-Stokes Flow Solver," AIAA-94-0061, 1994.

18. Kim, S.-E., Makarov, B. and Caraeni, D., "A Multi-Dimensional Linear Reconstruction Scheme for Arbitrary Unstructured Mesh," AIAA-2003-3990, 2003.
19. Delaney, K., Kim, S-E. and Shan, H., "Computational Investigation of Maneuvering Characteristics of Non-Bodies-of-Revolution," *Proceedings Open Source Computing International Conference (OSCIC)*, Munich, Germany, Nov., 2010.
20. Kim, S.-E. and Rhee, S. H., "Assessment of Eight Turbulence Models for a Three-Dimensional Boundary layer Involving Crossflow and Streamwise Vortices," AIAA Paper 2002-0852, 2002.
21. Kim, S.-E., Rhee, B., Shan, H., and Gorski, J., Paterson, E. and Maki, K., "Prediction of Turbulent Free-Surface Flows Around Surface Ships Using a Scalable Unstructured Finite-Volume Based RANS Solver" *Proceeding Gothenburg 2010 Workshop on CFD in Hydrodynamics*, Gothenburg, Sweden, Dec., 2010.
22. Shih, T.-H., Liou, W.W., Shabbir, A., and Zhu, J., "A New k-e Eddy-Viscosity Model for High Reynolds Number Turbulent Flows - Model Development and Validation," *Computers & Fluids*, Vol. 24, No. 3, pp. 227-238, 1995.
23. Menter, F. R., "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications," AIAA J., Vol.32, No.8, pp. 1598-1605, 1994.

Report Distribution

No of Copies Print	PDF	Office	Individual
	4	HPCMP	Paula Gibson, Myles Hurwitz, Richard Kendall, Doug Post
	2	PSU/ARL	David Boger, Eric Paterson
	1	Univ of Michigan	Kevin Maki
1		DTIC	

No of Copies Print	PDF	NSWCCD Code	Individual
	1	3452	Library
1		5060	D. Walden
	11	5700	P. Chang, K. Delaney, M. Ebert, J. Gorski, R. Miller, S-E. Kim, B. Rhee, H. Shan, J. Slomski, A. vonLoebbecke, W. Wilson
	1	5800	R. Hurwitz