Unstructured Finite Volume Method (FVM) – Incompressible & Compressible Flow Equations

Ashoke De

Air-force Chair Professor & Associate Dean of Academic Affairs

Computational Propulsion Lab, Dept. Of Aerospace Engineering, Dept. of Sustainable Energy Engineering, IITK

9-1-2024

NSM Workshop on
High **P**erformance **C**omputing
(HPC) **M**ethods for **C**omplex
and **M**oving **G**eometries
December 01-02, 2023

Part-I: Incompressible FVM methods in OF (Unstructured FVM)

Phases of CFD

Computational Fluid Dynamics [CFD]

Pre-Processing

- Problem Definition or Statement
- Geometrical and Physical Parameters
- Meshing/Discretization of the Geometry

Processing

- Modeling of PDEs as per the problem definition
- Use Finite Volume Method for the discretization
- Select different Numerical schemes for solution

Post-Processing

- Visualization of the obtained results
- Calculation of the derived quantities
- Comparison with the published results

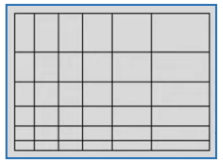
Pre-processing: CFD

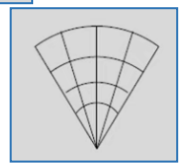
Problem Definition

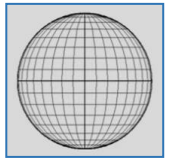
Geometrical and Physical Parameters

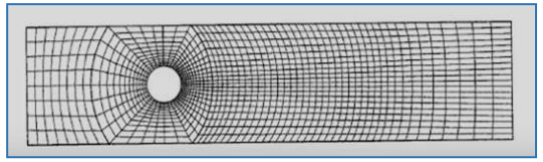
Grid / Meshing

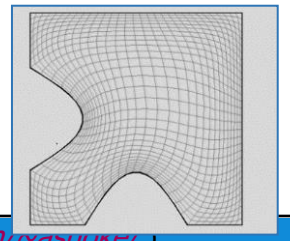
- Structured Grid
  - Regular Grid
    - Cartesian Grid
    - Cylindrical Grid
    - Spherical Grid
  - Block-Structured Grid
  - Curvilinear Grid



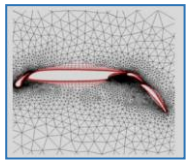


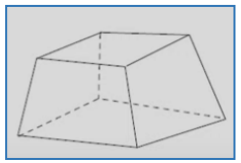


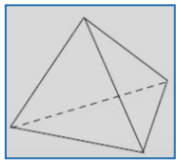


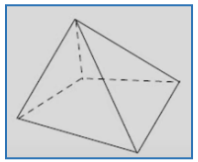


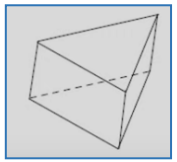
- Unstructured Grid
  - Regular Grid
    - Hexahedral
    - Tetrahedral
    - Pyramid
    - Prism
    - Triangular
  - Hybrid Grid

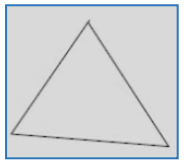


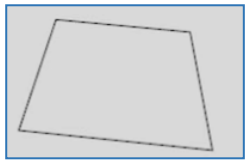






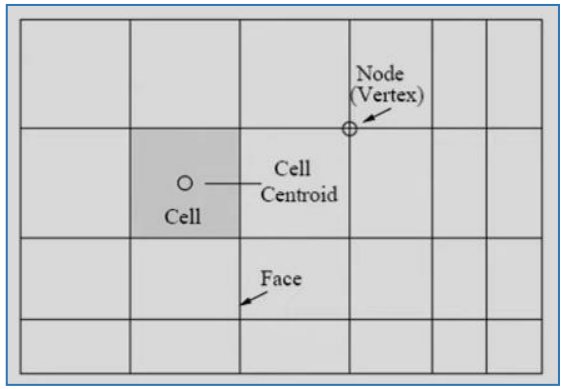




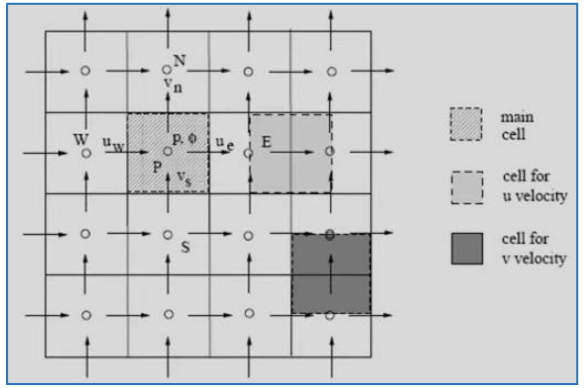


Pre-processing: CFD

Grid-Terminology

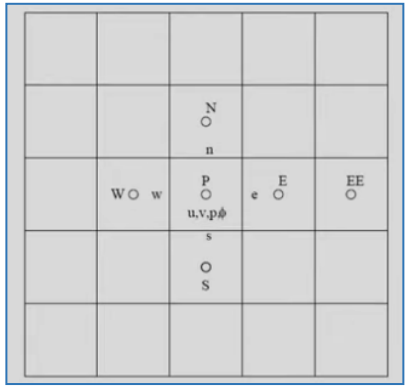


Staggered Grid



- ✓ Not all variable share the same grid points.
- ✓ Strong Coupling between Pressure and Velocities

Co-located Grid



- ✓ Store all variable at the same set of grid points
- ✓ Easy to code

Processing: CFD

Identification of the right approximations

- Steady/Unsteady?
- Laminar/Turbulent?
- Viscous/Inviscid?
- Incompressible/Compressible?
- Single Phase/Multi-phase?

Modeling the equations

- Continuity Equation
- Momentum Equation
- Energy Equation
- Equation of State
- Turbulence Equation

Solution of Equations

- Direct Methods
  - Gauss Elimination
  - LU Decomposition
  - Cholesky Method
  - Crout Method
- Iterative Methods
  - Point Jacobi
  - Gauss Seidel
  - S.O.R.
  - GMRES

Post-processing: CFD

Data Visualization

Calculation of derived quantities

ParaView
Parallel Visualization Application

MATLAB®

ORIGINPRO®
The Data Analysis and Graphing Workspace

R Studio

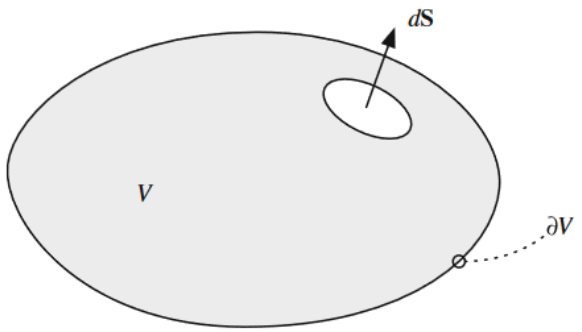
tecplot®

Excel

SIGMAPLOT®
Exact Graphs and Data Analysis

General Conservation Equation

Change of of ϕ over time Δt within the material volume (MV)	=	Surface flux of ϕ over time Δt across the control volume	+	Source/sink of ϕ over time Δt within control volume
Term I		Term II		Term III



$$\text{Term I} = \frac{d}{dt}\left(\int_{MV} (\rho\phi)dV\right) = \int_V \left[\frac{\partial}{\partial t}(\rho\phi) + \nabla\cdot(\rho\mathbf{v}\phi)\right]dV$$

$$\text{Term II} = -\int_S \mathbf{J}^{\phi}_{diffusion}\cdot\mathbf{n}\,dS = -\int_V \nabla\cdot\mathbf{J}^{\phi}_{diffusion}dV = \int_V \nabla\cdot(\Gamma^{\phi}\nabla\phi)dV$$

$$\text{Term III} = \int_V Q^{\phi}dV$$

$$\int_V \left[\frac{\partial}{\partial t}(\rho\phi) + \nabla\cdot(\rho\phi\mathbf{v})\right]dV = \int_V \nabla\cdot(\Gamma^{\phi}\nabla\phi)dV + \int_V Q^{\phi}dV$$

$$\int_V \left[\frac{\partial}{\partial t}(\rho\phi) + \nabla\cdot(\rho\mathbf{v}\phi) - \nabla\cdot(\Gamma^{\phi}\nabla\phi) - Q^{\phi}\right]dV = 0$$

$$\underbrace{\frac{\partial}{\partial t}(\rho\phi)}_{\textit{unsteady term}} + \underbrace{\nabla\cdot(\rho\mathbf{v}\phi)}_{\textit{convection term}} = \underbrace{\nabla\cdot(\Gamma^{\phi}\nabla\phi)}_{\textit{diffusion term}} + \underbrace{Q^{\phi}}_{\textit{source term}}$$

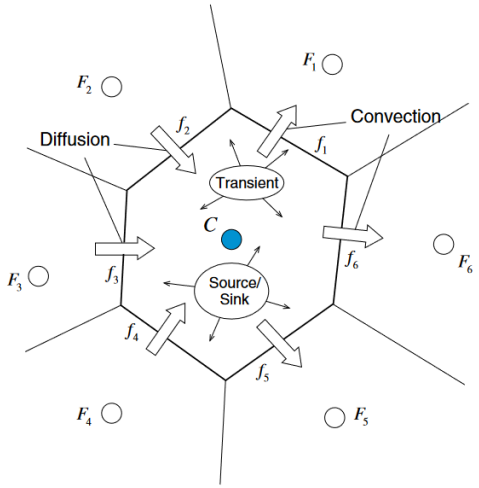
GEs/PDEs: CFD

General Conservation Equation

$$\underbrace{\frac{\partial}{\partial t}(\phi)}_{\textit{unsteady term}} + \underbrace{\nabla\cdot(\mathbf{v}\phi)}_{\textit{convection term}} = \underbrace{\nabla\cdot\left(D^{\phi}\nabla\phi\right)}_{\textit{diffusion term}} + \underbrace{P\phi - C}_{\textit{source and sink term}}$$

In OpenFOAM

```
(
    fvm::ddt(phi)
  + fvm::div(mDot, phi)
  - fvm::laplacian(Dphi, phi)
 ==
    fvm::Sp(P,phi)
   -fvc::(C)
);
```

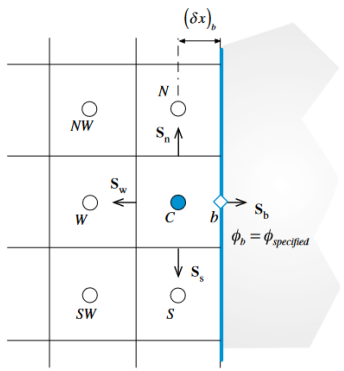


Objects	Type of data	OpenFOAM® Class
Interpolation	Differencing schemes	surfaceInterpolation<template>
Explicit discretization: differential operator	ddt, div, grad, curl	fvc::
Implicit discretization: differential operator	ddt, d2dt2, div, laplacian	fvm::

Boundary Conditions [Structured Mesh]

$$a_C\phi_C + a_W\phi_W + a_N\phi_N + a_S\phi_S = b_C$$

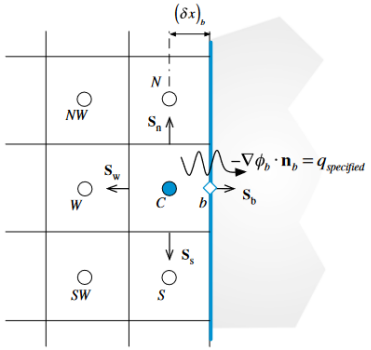
Dirichlet Boundary Condition



$$\phi_b = \phi_{specified}$$

$$a_E = 0$$
$$a_W = FluxF_w = -\Gamma_w^\phi gDiff_w$$
$$a_N = FluxF_n = -\Gamma_n^\phi gDiff_n$$
$$a_S = FluxF_s = -\Gamma_s^\phi gDiff_s$$
$$a_C = FluxC_b + \sum_{f\sim nb(C)} FluxC_f = FluxC_b + (FluxC_w + FluxC_n + FluxC_s)$$
$$b_C = Q_C^\phi V_C - \left(FluxV_b + \sum_{f\sim nb(C)} FluxV_f\right)$$

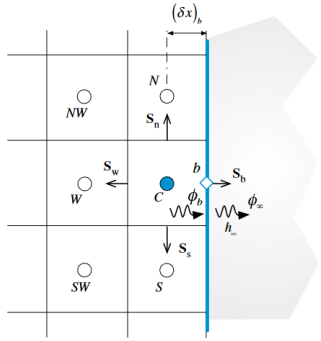
Von Neumann Boundary Condition



$$-\left(\Gamma^\phi \nabla\phi\right)_b \cdot \mathbf{i} = q_b$$

$$a_E = 0$$
$$a_W = FluxF_w = -\Gamma_w^\phi gDiff_w$$
$$a_N = FluxF_n = -\Gamma_n^\phi gDiff_n$$
$$a_S = FluxF_s = -\Gamma_s^\phi gDiff_s$$
$$a_C = \sum_{f\sim nb(C)} FluxC_f = -(a_W + a_N + a_S)$$
$$b_C = Q_C^\phi V_C - \left(FluxV_b + \sum_{f\sim nb(C)} FluxV_f\right)$$

Mixed Boundary Condition



$$\mathbf{J}_b^{\phi,D} \cdot \mathbf{S}_b = -\left(\Gamma^\phi \nabla\phi\right)_b \cdot \mathbf{i}S_b = -h_\infty(\phi_\infty - \phi_b)(\Delta y)_C$$

$$a_E = 0$$
$$a_W = FluxF_w = -\Gamma_w^\phi gDiff_w$$
$$a_N = FluxF_n = -\Gamma_n^\phi gDiff_n$$
$$a_S = FluxF_s = -\Gamma_s^\phi gDiff_s$$
$$a_C = FluxC_b + \sum_{f\sim nb(C)} FluxC_f = (FluxC_b + FluxC_w + FluxC_n + FluxC_s)$$
$$b_C = Q_C^\phi V_C - \left(FluxV_b + \sum_{f=nb(C)} FluxV_f\right)$$

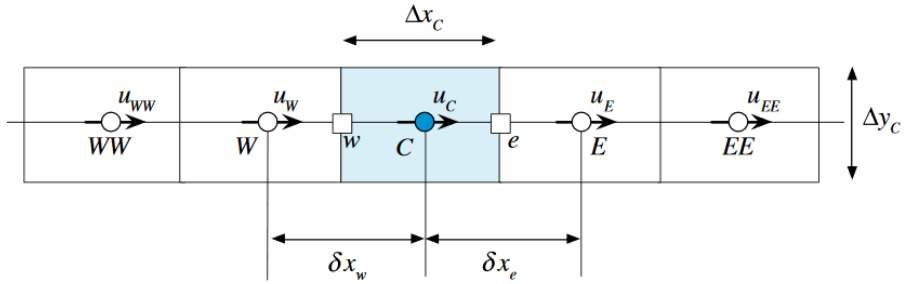
Incompressible Flow Eqns: CFD

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0$$

$$\frac{\partial}{\partial t}[\rho \mathbf{v}] + \nabla \cdot \{\rho \mathbf{v}\mathbf{v}\} = -\nabla p + \nabla \cdot \left\{\mu\left[\nabla \mathbf{v} + (\nabla \mathbf{v})^{\mathrm{T}}\right]\right\} + \mathbf{f}_b$$

- ✓ Non-linear equation
- ✓ The **stress tensor** can be reformulated into a **diffusion-like term** and treated implicitly
- ✓ The **velocity** field can be computed using the momentum equation
- ✓ The **pressure** field appearing in the momentum equation cannot be computed directly from the continuity equation
- ✓ Consequently, an equation for pressure is required and should be derived

Discretization of Incompressible Eqs: CFD



Discretization of Momentum Equation

$$\frac{\partial(\rho u u)}{\partial x} = \frac{\partial}{\partial x}\left(\mu \frac{\partial u}{\partial x}\right) - \frac{\partial p}{\partial x}$$

$$\int_{V_C} \frac{\partial(\rho u u)}{\partial x} dV = \int_{V_C} \frac{\partial}{\partial x}\left(\mu \frac{\partial u}{\partial x}\right) dV - \int_{V_C} \frac{\partial p}{\partial x} dV$$

Using Divergence theorem

$$\int_{\partial V_C} (\rho u u) dy = \int_{\partial V_C} \mu \frac{\partial u}{\partial x} dy - \int_{V_C} \frac{\partial p}{\partial x} dV$$

Representing the surface integrals by summation of fluxes over the faces of the element, and using a single Gaussian point for the face integrals

$$\underbrace{(\rho u \Delta y)_e}_{\dot{m}_e} u_e + \underbrace{-(\rho u \Delta y)_w}_{\dot{m}_w} u_w = \left(\mu \frac{\partial u}{\partial x} \Delta y\right)_e - \left(\mu \frac{\partial u}{\partial x} \Delta y\right)_w - \int_{V_C} \frac{\partial p}{\partial x} dV$$

$$\underbrace{\dot{m}_e u_e + \dot{m}_w u_w}_{Convection} - \underbrace{\left[\left(\mu \frac{\partial u}{\partial x} \Delta y\right)_e - \left(\mu \frac{\partial u}{\partial x} \Delta y\right)_w\right]}_{Diffusion} = -\int_{V_C} \frac{\partial p}{\partial x} dV$$

$$\boxed{a_C^u u_C + \sum_{F \sim NB(C)} (a_F^u u_F) = b_C^u - \int_{V_C} \frac{\partial p}{\partial x} dV}$$

Discretization of Continuity Equation

$$\frac{\partial(\rho u)}{\partial x} = 0$$

$$\int_{V_C} \frac{\partial(\rho u)}{\partial x} dV = 0$$

Using Divergence theorem

$$\sum_{f \sim nb(C)} (\rho u \Delta y)_f = (\rho u \Delta y)_e - (\rho u \Delta y)_w = 0$$

$$\boxed{\sum_{f \sim nb(C)} \dot{m}_f = \dot{m}_e + \dot{m}_w = 0}$$

I am curious about OpenFOAM … but which version?

OpenFOAM®

openfoam.com

OpenFOAM

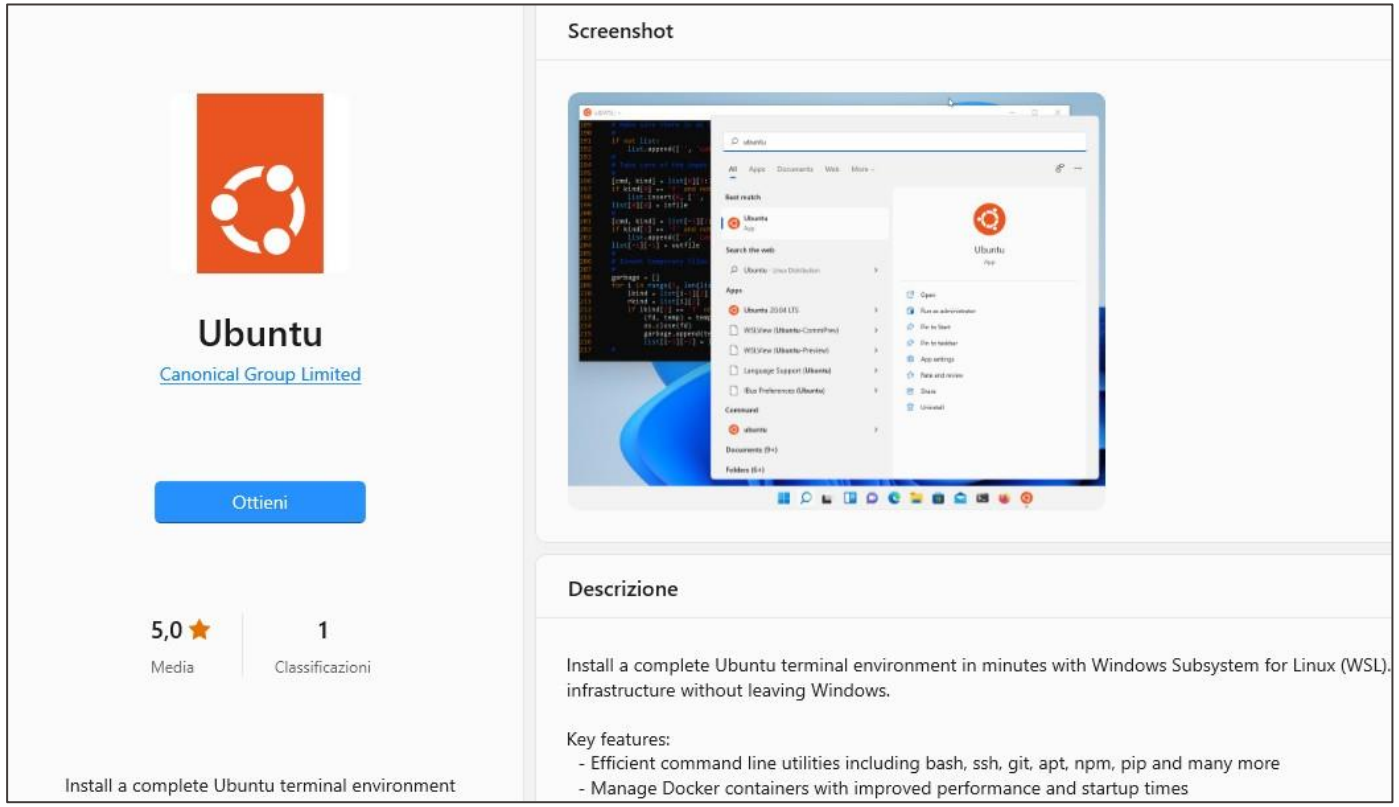
openfoam.org

IMPORTANT!

If you want to use an available solver, or take features from available solvers for your own solver, be very careful and select the right OF version!

Can I use it on my computer?

OpenFOAM runs natively on Linux systems…



MAC, or the Linux subsystem for Windows can be used, but **not recommended by the presenter**

How to get OpenFOAM?

Follow the simple steps on the download page

(example for OF-9 from the .org version)

Installation

OpenFOAM and *ParaView* can be simply installed for the first time using the **apt** package management tool. The user will need to provide superuser password authentication when executing the following commands with **sudo**

1. **Copy and paste** the following in a **terminal prompt** (*Applications → Accessories → Terminal*) to add **dl.openfoam.org** to the list of software repositories for **apt** to search, and to add the public key (**gpg.key**) for the repository to enable package signatures to be verified.
   Note: use secure **https://** **for the public key** to ensure secure transfer, but use **http://** **for the repository**, since **https://** may not be supported and is not required since the key provides secure authentication of the package files.

```
sudo sh -c "wget -O - https://dl.openfoam.org/gpg.key | apt-key add -"
sudo add-apt-repository http://dl.openfoam.org/ubuntu
```

   **Note: This only needs to be done once for a given system

2. Update the **apt** package list to account for the new download repository location

```
sudo apt-get update
```

3. Install OpenFOAM (9 in the name refers to version 9) which also installs **paraviewopenfoam56** as a dependency.

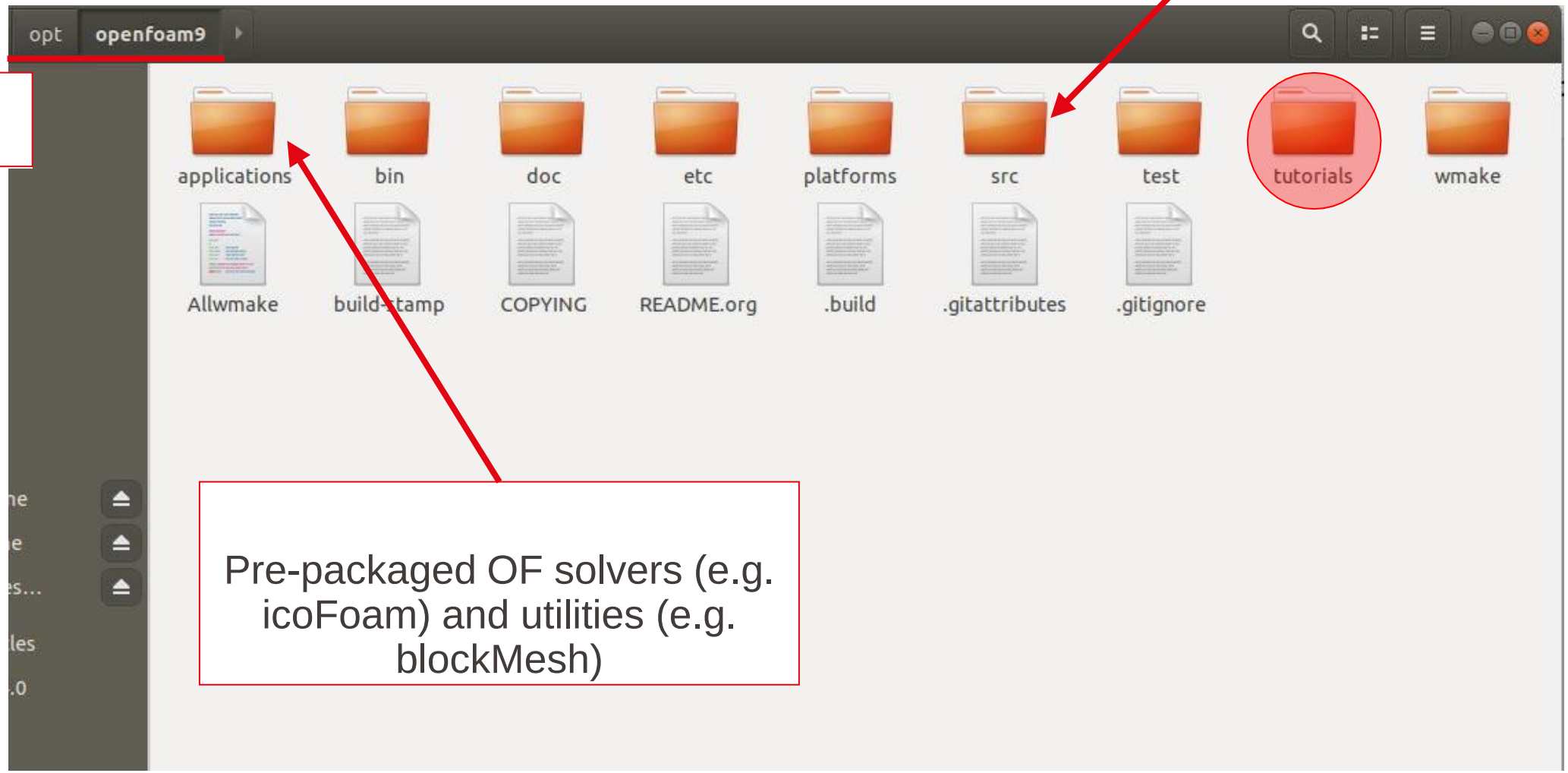
```
sudo apt-get -y install openfoam9
```

OpenFOAM 9 and *ParaView 5.6.3* are now installed in the */opt* directory.

What comes with OpenFOAM?

Main OF library

Typical location



Pre-packaged OF solvers (e.g. icoFoam) and utilities (e.g. blockMesh)

Learn OpenFOAM - Official documentation

- https://cfd.direct/openfoam/user-guide/
- https://www.openfoam.com/documentatio n/user-guide

It includes some post-processing examples

CFD Direct
The Architects of OpenFOAM

Home Book OpenFOAM Cloud

OpenFOAM v9 User Guide: 2 Tutorials

[Table of Contents] [Index] [Version 9 | Version 8 | Version 7 | Version 6 | Version 5 | Version 4]

[prev] [next]

Chapter 2 Tutorials

In this chapter we shall describe in detail the process of setup, simulation and post-processing for some OpenFOAM test cases, with the principal aim of introducing a user to the basic procedures of running OpenFOAM. The $FOAM_TUTORIALS directory contains many more cases that demonstrate the use of all the solvers and many utilities supplied with OpenFOAM.

Before attempting to run the tutorials, the user must first make sure that OpenFOAM is installed correctly. Cases in the tutorials will be copied into the so-called run directory, an OpenFOAM project directory in the user's file system at $HOME/OpenFOAM/<USER>-6/run where <USER> is the account login name and "6" is the OpenFOAM version number. The run directory is represented by the $FOAM_RUN environment variable enabling the user to check its existence conveniently by typing

```
ls $FOAM_RUN
```

If a message is returned saying no such directory exists, the user should create the directory by typing

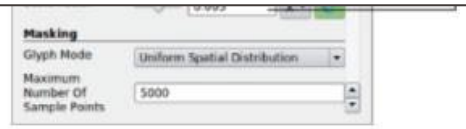


Figure 2.7: Properties panel for the Glyph filter.

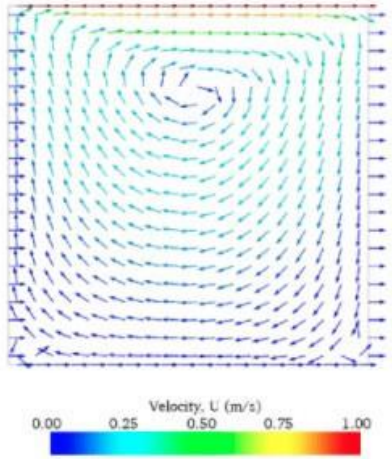


Figure 2.8: Velocities in the cavity case.

Learn OpenFOAM - Browse the C++ source guide official documentation

- https://www.openfoam.com/documentation/guides/v2112/doc/
- https://cpp.openfoam.org/v9/

- fixedGradientFvPatchField
- fixedInternalValueFvPatchField
- fixedJumpAMIFvPatchField
- fixedJumpFvPatchField
- FixedList
- fixedMeanFvPatchField
- fixedMeanOutletInletFvPatchField
- fixedMultiPhaseHeatFluxFvPatchScalarField
- fixedNormalInletOutletVelocityFvPatchVectorField
- fixedNormalSlipFvPatchField
- fixedNormalSlipPointPatchField
- fixedPressureCompressibleDensityFvPatchScalarField
- fixedProfileFvPatchField
- fixedRhoFvPatchScalarField
- fixedShearStressFvPatchVectorField
- fixedTrim
- fixedUnburntEnthalpyFvPatchScalarField
- fixedValueFvPatchField
- fixedValueFvsPatchField
- fixedValuePointPatchField
- flipLabelOp
- flipOp
- flowRateInletVelocityFvPatchVectorField
- flowRateOutletVelocityFvPatchVectorField
- fluentFvMesh
- fluidReactionThermo
- fluidSolutionControl

Detailed Description

template<class Type>
class Foam::fixedGradientFvPatchField< Type >

This boundary condition supplies a fixed gradient condition, such that the patch values are calculated using:

$$x_p = x_c + \frac{\nabla(x)}{\Delta}$$

where

x_p	=	patch values
x_c	=	internal field values
$\nabla(x)$	=	gradient (user-specified)
Δ	=	inverse distance from patch face centre to cell centre

Usage

Property	Description	Required	Default value
gradient	gradient	yes	

Example of the boundary condition specification:

```
<patchName>
{
    type            fixedGradient;
    gradient        uniform 0;
}
```

Learn OpenFOAM - Plenty of additional resources

- Tutorials/lectures (have a look on Google or YouTube)
- Master/PhD thesis etc.
- Forums
- (Often) direct communication with solver developers

And remember:

- **Don't get frustrated: there is always a way out with OpenFOAM and, most likely, someone who had your same problem and will be happy to help**
- **Don't get discouraged: the entry barrier may seem steep, but skills you'll learn will allow you to tackle any kind of problems**
- **If possible, do not do it alone!**

OpenFOAM® brief overview

General description:

- OpenFOAM® stands for Open Source Field Operation and Manipulation.
- OpenFOAM® is first and foremost a C++ library used to solve partial differential equations (PDEs), and ordinary differential equations (ODEs).
- It comes with several ready-to-use or out-of-the-box solvers, pre-processing utilities, and post-processing utilities.
- It is licensed under the GNU General Public License (GPL). That means it is freely available and distributed with the source code.
- It can be used in massively parallel computers. No need to pay for separate licenses.
- It is under active development.
- It counts with a wide-spread community around the world (industry, academia and research labs).

OpenFOAM® brief overview

Multi-physics simulation capabilities:

- OpenFOAM® has extensive multi-physics simulation capabilities, among others:
  - Computational fluid dynamics (incompressible and compressible flows).
  - Computational heat transfer and conjugate heat transfer.
  - Combustion and chemical reactions.
  - Multiphase flows and mass transfer.
  - Particle methods (DEM, DSMC, MD) and lagrangian particles tracking.
  - Stress analysis and fluid-structure interaction.
  - Rotating frames of reference, arbitrary mesh interface, dynamic mesh handling, and adaptive mesh refinement.
  - 6 DOF solvers, ODE solvers, computational aero-acoustics, computational electromagnetics, computational solid mechanics, MHD.

Physical modeling library:

❑OpenFOAM® comes with many physical models, among others:

- Extensive turbulence modeling capabilities (RANS, DES and LES).
- Transport/rheology models. Newtonian and non-Newtonian viscosity models.
- Thermophysical models and physical properties for liquids and gases.
- Source terms models.
- Lagrangian particle models.
- Interphase momentum transfer models for multiphase flows.
- Combustion, flame speed, chemical reactions, porous media, radiation, phase change.

OpenFOAM® brief overview

Under the hood you will find the following:

- Finite Volume Method (FVM) based solver.
- Collocated polyhedral unstructured meshes.
- Second order accuracy in space and time. Many discretization schemes available (including high order methods).
- Steady and transient solvers available.
- Pressure-velocity coupling via segregated methods (SIMPLE and PISO).
- But coupled solvers are under active development.
- Massive parallelism through domain decomposition.
- It comes with its own mesh generation tools.
- It also comes with many mesh manipulation and conversion utilities.
- It comes with many post-processing utilities.
- All components implemented in library form for easy re-use.

OpenFOAM® vs. Commercial CFD applications:

✓ OpenFOAM® capabilities mirror those of commercial CFD applications.

✓ The main differences with commercial CFD applications are:

- There is no native GUI.
- It does not come with predefined setups. The users need to have a basic understanding of the CFD basics and be familiar with OpenFOAM® command line interface (CLI).
- Knowing your way around the Linux bash shell is extremely useful.
- It is not a single executable. Depending of what you are looking for, you will need to execute a specific application from the CLI.
- It is not well documented, but the source code is available.
- Access to complete source = no black magic. But to understand the source code you need to know object-oriented programming and C++.
- Solvers can be tailored for a specific need, therefore OpenFOAM® is ideal for research and development.
- It is free and has no limitation on the number of cores you can use.

Developing new solvers (in case you need it):

•As the user has complete access to the source code, she/he has total freedom to modify existing solvers or use them as the starting point for new solvers.

•New solvers can be easily implemented using OpenFOAM® high level programming, *e.g.*:

$$\frac{\partial T}{\partial t} + \nabla \cdot (\phi T) - \nabla \cdot (\nu \nabla T) = 0 \longrightarrow$$

```
solve
(
fvm::ddt(T)
+ fvm::div(phi,T)
- fvm::laplacian(nu,T)
== 0
);
```

Correspondence between the implementation and the original equation is clear.

OpenFOAM® directory organization

```
$WM_PROJECT_DIR
├── Allwmake
├── applications
├── bin
├── COPYING
├── doc
├── etc
├── platforms
├── README.org
├── src
├── tutorials
└── wmake
```

If you installed OpenFOAM® in the default location, the environment variable **`$WM_PROJECT_DIR`** should point to the following directory (depending on the installed version):

`$HOME/OpenFOAM/OpenFOAM-8`

or

`$HOME/OpenFOAM/OpenFOAM-dev`

In this directory you will find all the files containing OpenFOAM® installation.

In this directory you will also find additional files (such as *`README.org`*, *`COPYING`*, etc.), but the most important one is *`Allwmake`*, which compiles OpenFOAM®.

OpenFOAM® - applications/utilities

- You will find all the applications in the directory **`$FOAM_SOLVERS`** (you can use the alias `sol` to go there).
- You will find all the utilities in the directory **`$FOAM_UTILITIES`** (you can use the alias `util` to go there).
- For example, in the directory **`$FOAM_SOLVERS`**, you will find the directories containing the source code for the solvers available in the OpenFOAM® installation:

- **basic**
- **combustion**
- **compressible**
- **discreteMethods**
- **DNS**
- **electromagnetics**
- **financial**
- **heatTransfer**
- **incompressible**
- **lagrangian**
- **multiphase**
- **stressAnalysis**

- The solvers are subdivided according to the physics they address.
- The utilities are also subdivided in a similar way.

OpenFOAM® - applications/utilities

- For example, in the sub-directory **`incompressible`** you will find several sub-directories containing the source code of the following solvers:

  - **adjointShapeOptimizationFoam**
  - **boundaryFoam**
  - **icoFoam**
  - **nonNewtonianIcoFoam**
  - **pimpleFoam**
  - **pisoFoam**
  - **shallowWaterFoam**
  - **simpleFoam**

- Inside each directory, you will find a file with the extension `*.C` and the same name as the directory. This is the main file, where you will find the top-level source code and a short description of the solver or utility.
- For example, in the file `incompressible/icoFoam/icoFoam.C` you will find the following description:

Transient solver for incompressible, laminar flow of Newtonian fluids.

OpenFOAM® - applications/utilities

- ✓ Remember, OpenFOAM® is not a single executable.
- ✓ You will need to find the solver or utility that best fit what you want to do.
- ✓ A few solvers that we will use during this course:
  - **icoFoam**: laminar incompressible unsteady solver. Be careful, do not use this solver for production runs as it has many limitations.
  - **simpleFoam**: incompressible steady solver for laminar/turbulent flows.
  - **pimpleFoam**: incompressible unsteady solver for laminar/turbulent flows.
  - **rhoSimpleFoam**: compressible steady solver for laminar/turbulent flows.
  - **rhoPimpleFoam**: unsteady compressible solver for (laminar/turbulent flows.
  - **interFoam**: unsteady multiphase solver for separated flows using the VOF method (laminar and turbulent flows).
  - **laplacianFoam**: Laplace equation solver.
  - **potentialFoam**: potential flow solver.
  - **scalarTransportFoam**: steady/unsteady general transport equation solver.

OpenFOAM® - applications/utilities

- Take your time and explore the source code.
- Also, while exploring the source code be careful not to add unwanted modifications in the original installation.
- If you modify the source code, be sure to do the modifications in your user directory instead of the main source code.

Incompressible Flow Eqns in OF: CFD

Incompressible Flow Solution in OpenFOAM solvers[icoFoam]

- In OpenFOAM, the convection velocities is defined on the cell faces (phi) and the pressure is actually a pressure divided by a density

$$\frac{\partial U}{\partial t} + \nabla \cdot (\emptyset U) - \nabla \cdot (\mu \nabla U) = -\nabla p$$

- To develop the pressure equation, we write the previous equation in a semi-discretized form (implicit Euler):

$$\vartheta \frac{U_P^{n+1} - U_P^n}{\delta t} = \boxed{-a'_P U_P^{n+1} + \sum_{NP} a'_{NP} U_{NP}^{n+1}} - \nabla p$$

Discretization of the convective and the diffusive terms

- It can be recast into: $\left(\frac{\vartheta}{\delta t} + a'_P\right) U_P^{n+1} = \sum_{NP} a'_{NP} U_{NP}^{n+1} + \frac{\vartheta}{\delta t} U_P^n - \nabla p$
- Or $a_P U_P = H(U) - \nabla p$

Diagonal coefficients of the matrix

Contains the off-diagonal coefficients and the source terms (body forces + half of the discretization of the transient term)

- Or, $U_P = \frac{1}{a_P} H(U) - \frac{1}{a_P} \nabla p$
- Combining this equation with continuity equation leads to: $\nabla \cdot \left(\frac{1}{a_P} \nabla p\right) = \nabla \cdot \left(\frac{H(U)}{a_P}\right)$
- In this equation, a_P and H(U) are evaluated from the velocity field of the previous step.

Incompressible Flow Eqns in OF: CFD

Incompressible Flow Solver [icoFoam]

Solves the incompressible Navier-Stokes Equation:

$$\nabla.\mathbf{u} = 0$$

$$\frac{\partial u_x}{\partial t} + \nabla.(\mathbf{u}u_x) = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\nabla^2 u_x$$

$$\frac{\partial u_y}{\partial t} + \nabla.(\mathbf{u}u_y) = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\nabla^2 u_y$$

$$\frac{\partial u_z}{\partial t} + \nabla.(\mathbf{u}u_z) = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\nabla^2 u_z$$

Uses **PISO** (pressure-implicit splitting operator) algorithm.

- ✓ Guess p, flux (from previous time step)
- ✓ Velocity predictor : solve momentum equations to get U_x, U_y, U_z
- ✓ Solve pressure equation $\nabla \cdot \left[\frac{1}{A}\Delta p\right] = (flux\ term)$
- ✓ Correct flux to satisfy continuity
- ✓ go back to 2. until convergence is reached

icoFoam.C

```
Info<< "\nStarting time loop\n" << endl;

while (runTime.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

    #include "CourantNo.H"

    // Momentum predictor

    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
      + fvm::div(phi, U)
      - fvm::laplacian(nu, U)
    );

    if (piso.momentumPredictor())
    {
        solve(UEqn == -fvc::grad(p));
    }
```

Time loop starts

CFL no. is calculated & printed on screen

Momentum equation is Defined

U* is predicted from the pressure field of the previous time step

Incompressible Flow Eqns in OF: CFD

Incompressible Flow Solver [icoFoam]

icoFoam.C

```
// --- PISO loop
while (piso.correct())
{
    volScalarField rAU(1.0/UEqn.A());
    volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::flux(HbyA)
      + fvc::interpolate(rAU)*fvc::ddtCorr(U, phi)
    );

    adjustPhi(phiHbyA, U, p);

    // Update the pressure BCs to ensure flux consistency
    constrainPressure(p, U, phiHbyA, rAU);

    // Non-orthogonal pressure corrector loop
    while (piso.correctNonOrthogonal())
    {
        // Pressure corrector

        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAU, p) == fvc::div(phiHbyA)
        );

        pEqn.setReference(pRefCell, pRefValue);

        pEqn.solve(mesh.solver(p.select(piso.finalInnerIter())));

        if (piso.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }

    #include "continuityErrs.H"

    U = HbyA - rAU*fvc::grad(p);
    U.correctBoundaryConditions();
}
```

Update of a_P from U freshly computed

Update of a_P/H from U freshly computed

Projection of a_P/H over the cell faces

Insure mass conservation by adjusting the in-coming and out-coming flux if the BC are ill-defined

Pressure equation $\nabla \cdot \left(\frac{1}{a_P}\nabla p\right) = \nabla \cdot \left(\frac{H(U)}{a_P}\right)$ is defined

Pressure equation is solved

Here, we recover the right value of the flux velocity, $\emptyset = \frac{H(U)}{a_P} \cdot S - \frac{1}{a_P}\nabla p \cdot S$

Print the continuity error on screen

Velocity corrector stage: $U_P = \frac{1}{a_P}H(U) - \frac{1}{a_P}\nabla p$

Evaluate the velocity fields

$$\nabla.\mathbf{u} = 0$$

$$\frac{\partial u_x}{\partial t} + \nabla.(\mathbf{u}u_x) = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\nabla^2 u_x$$

$$\frac{\partial u_y}{\partial t} + \nabla.(\mathbf{u}u_y) = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\nabla^2 u_y$$

$$\frac{\partial u_z}{\partial t} + \nabla.(\mathbf{u}u_z) = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\nabla^2 u_z$$

System/fvScheme

```
ddtSchemes
{
    default         Euler;
}

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

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

laplacianSchemes
{
    default         Gauss linear orthogonal;
}
```

Directory structure of a general case

```
case_name
├── 0
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── system
│   ├── controlDict
│   ├── fvSchemes
│   └── fvSolution
└── time_directories
```

- OpenFOAM® uses a very particular directory structure for running cases.
- You should always follow the directory structure, otherwise, OpenFOAM® will complain.
- To keep everything in order, the case directory is often located in the path **`$WM_PROJECT_USER_DIR/run`**.
- This is not compulsory but highly advisable. You can copy the case files anywhere you want.
- The name of the case directory is given by the user (**do not use white spaces or strange symbols**).
- Depending of the solver or application you would like to use, you will need different files in each sub-directory.
- Remember, you always run the applications and utilities in the **top level** of the case directory (**the directory with the name case_name**). Not in the directory **`system`**, not in the directory **`constant`**, not in the directory **`0`**.

Directory structure of a general case

```
case_name
├── 0
│   ├── p
│   └── U
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── system
│   ├── controlDict
│   ├── fvSchemes
│   └── fvSolution
└── time_directories
```

`case_name`: the name of the case directory is given by the user (**do not use white spaces or strange symbols**).

This is the top-level directory, where you run the applications and utilities.

`system`: contains run-time control and solver numerics.

`constant`: contains physical properties, turbulence modeling properties, advanced physics and so on.

`constant/polyMesh`: contains the polyhedral mesh information.

`0`: contains boundary conditions (BC) and initial conditions (IC).

`time_directories`: contains the solution and derived fields. These directories are created by the solver automatically and according to the preset saving frequency, *e.g.*, 1, 2, 3, 4, … , 100.

Before we start – Always remember the directory structure

```
case_name
├── 0
├── constant
│   └── polyMesh
├── system
└── time_directories
```

- To keep everything in order, the case directory is often located in the path **`$WM_PROJECT_USER_DIR/run`**.
- This is not compulsory but highly advisable, you can put the case in any directory of your preference.
- The name of the case directory if given by the user (do not use white spaces).
- You run the applications and utilities in the top level of this directory.
- The directory **`system`** contains run-time control and solver numerics.
- The directory **`constant`** contains physical properties, turbulence modeling properties, advanced physics and so on.
- The directory **`constant/polyMesh`** contains the polyhedral mesh information.
- The directory **`0`** contains boundary conditions (BC) and initial conditions (IC).

Incompressible Flow Eqns in OF: CFD

Incompressible Flow Solver [icoFoam]

System/fvSolution

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

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

Linear-solver to solve the system of discretized equations

PISO algorithm controls

System/controlDict

```
application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable true;
```

Select Solver name

Initial time

Final time

Time-step or Delta T

Writing interval

Writing format

Writing precision

Demo examples in OF: CFD

[Incompressible Flow]

Problem Set - I

Backward Step

Type: Unsteady, Incompressible
Flow: Laminar/Turbulent
Solver: icoFoam/pimpleFoam

Problem Set - II

Oscillating Cylinder

Type: Unsteady, Moving Mesh
Flow: Laminar
Solver: pisoFoam/pimpleFoam

Multiphase **F**low **E**quations
[*Pressure based*]

Incompressible Multiphase Flow Equations (interIsoFoam solver)

- ✓ The governing equations for flows consisting of **two immiscible, incompressible fluids**:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = -\nabla \cdot p + \nabla \cdot \boldsymbol{T} + \rho \boldsymbol{f} + f_\sigma$$

- ✓ where $\boldsymbol{\rho}$ is the density field, **p** is the pressure, **T** is the stress tensor, **f** represents the body forces, and $\mathbf{f_\sigma}$ is the surface tension.
- ✓ The governing equations will take a ”**one fluid**” approach, where the two fluids are modelled using one common density field with a sharp jump at the interface. (Tryggvason et al.,2011).
- ✓ The VOF method is an Eulerian volume tracking method where a step function is used to mark the location of the phases (water and air).

Incompressible Multiphase Flow Equations (interIsoFoam solver)

- ✓ The VOF method adds one governing equation for the transport of the volume fraction α, the advection equation:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{u}) = 0$$

- ✓ The step function α marks the volume fraction of the tracked phase in a control volume, so that α = 1 corresponds to a control volume entirely occupied by water and α = 0 corresponds to a control volume not containing any water, only air. The value of α is averaged in each of the mesh cells. The interface between the phases is found in cells where 0 < α < 1.

$$\alpha = \begin{cases} 0 & phase\ 2\ (\text{air}) \\ 0 < \alpha < 1 & interface \\ 1 & phase\ 1\ (water) \end{cases}$$

$$\rho = \alpha \rho_{water} + (1-\alpha)\rho_{air}$$

$$\mu = \alpha \mu_{water} + (1-\alpha)\mu_{air}$$

- ✓ The main drawback of the VOF method is the smearing of the water surface. Without any additional surface capturing method the surface will be represented as a region where α gradually changes from 1 to 0.

Incompressible Multiphase Flow Equations (interIsoFoam solver)

- ✓ Interface reconstruction for the liquid-gas interface is another major process in these multiphase solvers. ANSYS Fluent uses the **geometric reconstruction scheme**, *interFoam* uses a scheme called **MULES** for improving the surface sharpness and the newly developed method **isoAdvector** (Roenby et al., (2016)) was introduced, with the solver *interIsoFoam*.
- ✓ Non-linear equation
- ✓ The **stress tensor** can be reformulated into a **diffusion-like term** and treated implicitly
- ✓ The **velocity** field can be computed using the momentum equation
- ✓ The **pressure** field appearing in the momentum equation cannot be computed directly from the continuity equation
- ✓ Consequently, an equation for pressure is required and should be derived

Advection of the interface: isoAdvector

The **isoAdvector** algorithm (Roenby et al., (2016)) is a **geometric volume-of-fluid** (VoF) method. That is, we explicitly reconstruct the interface between the fluids and then advect the interface to have better estimates of the volume fluxes.

- Integral form of conservation of mass reduces to passive advection of the Heaviside function(H):

$$\int_C \frac{\partial \rho}{\partial t}\,\mathrm{d}V + \int_C \nabla\cdot(\rho\mathbf{U})\,\mathrm{d}V = 0 \qquad \int_C \frac{\partial H}{\partial t}\,\mathrm{d}V + \int_C \nabla\cdot(H\mathbf{U})\,\mathrm{d}V = 0$$

- Considering **Leibniz integral rule** for the transient term and the divergence theorem for the advection term:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_C H\,\mathrm{d}V + \int_{\partial C} H\mathbf{U}\cdot\mathbf{n}\,\mathrm{d}A = 0$$

- By introducing the volume fraction of the reference fluid (here the heavy fluid) in a cell and integrating forward in time (from time $t = t_n$ to time $t = t_{n+1}$):

$$\alpha_C(t) = \frac{1}{|V_C|}\int_C H\,\mathrm{d}V \qquad \alpha_C(t^{n+1}) = \alpha_C(t^n) - \frac{1}{|V_C|}\sum_f \int_{t^n}^{t^{n+1}} \int_f H\mathbf{U}\cdot\mathbf{n}\,\mathrm{d}A\,\mathrm{d}\tau$$

- The purpose of the isoAdvector algorithm is then to approximate the above **double integral** in a Lagrangian manner:

$$\Delta V_f = \int_{t^n}^{t^{n+1}} \int_f H\mathbf{U}\cdot\mathbf{n}\,\mathrm{d}A\,\mathrm{d}\tau$$

Volumetric face flux: $\phi_f(t) = \int_f \mathbf{U}\cdot\mathbf{n}\,\mathrm{d}A$

- $|\mathbf{S}_f|$ denotes the magnitude of the surface normal vector, i.e. the area of the face and approximated as:

$$\Delta V_f \approx \int_{t^n}^{t^{n+1}} \frac{\phi_f(\tau)}{|\mathbf{S}_f|}\int_f H\,\mathrm{d}A\,\mathrm{d}\tau \approx \frac{\phi_f^n}{|\mathbf{S}_f|}\int_{t^n}^{t^{n+1}} \int_f H\,\mathrm{d}A\,\mathrm{d}\tau$$

- With the evaluation of ΔVf *(dVf_)* at all the downwind faces of the interface cells, the actual **advection** is performed. Updated volume fraction is then calculated as:

$$\alpha_C(t^{n+1}) = \alpha_C(t^n) - \frac{1}{|V_C|}\sum_f \Delta V_f$$

Incompressible Multiphase Flow Equations (interIsoFoam solver)

Using PIMPLE Algorithm

- An equation for the pressure using the incompressibility condition is derived.
- This pressure is then used to correct an auxiliary velocity field thus making it incompressible.
- $\mathbf{U}_f$ = Face value of the convected velocity
- m =latest update of the velocity field (from the inner loop)
- C = current cell and N = neighbour cell
- LHS of the discretized momentum equation: ⟶
- **Auxiliary velocity:** U*c
- Adding the gravitational force, the auxiliary velocity field:

$$\int_C \frac{\partial(\rho\mathbf{U})}{\partial t}\,\mathrm{d}V + \int_C \nabla\cdot(\rho\mathbf{U}\mathbf{U})\,\mathrm{d}V = -\int_C \nabla p_d\,\mathrm{d}V - \int_C (\mathbf{g}\cdot\mathbf{x})\nabla\rho\,\mathrm{d}V$$

$$\int_C \nabla\cdot\mathbf{U}\,\mathrm{d}V = 0 \qquad \int_C \frac{\partial(\rho\mathbf{U})}{\partial t}\,\mathrm{d}V = \frac{\mathrm{d}}{\mathrm{d}t}\int_C \rho\mathbf{U}\,\mathrm{d}V$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_C \rho\mathbf{U}\,\mathrm{d}V \approx \frac{\langle\rho\mathbf{U}\rangle_C^{n+1} - \langle\rho\mathbf{U}\rangle_C^{n}}{\Delta t}|V_C|$$

$$\mathbf{U}_f = \langle\mathbf{U}\rangle_C^{n+1} w_C + \langle\mathbf{U}\rangle_N^{m} w_N$$

$$\boxed{\frac{\langle\rho\rangle_C^{n+1}\langle\mathbf{U}^*\rangle_C - \langle\rho\mathbf{U}\rangle_C^{n}}{\Delta t}|V_C| + \sum_f \left(\langle\mathbf{U}\rangle_C^{*} w_C + \langle\mathbf{U}\rangle_N^{m} w_N\right)\left([\rho]\frac{\Delta V_f}{\Delta t} + \rho^{-}\phi_f^{n}\right) = \ldots}$$

$$\boxed{\langle\mathbf{U}^*\rangle_C = \frac{\mathcal{H}\left(\langle\mathbf{U}\rangle_N^{m}\right)}{a_C} - \frac{\langle(\mathbf{g}\cdot\mathbf{x})\nabla\rho\rangle_C^{n+1}}{a_C}}$$

$$\mathcal{H}\left(\langle\mathbf{U}\rangle_N^{m}\right) = \sum_f a_N\langle\mathbf{U}\rangle_N^{m} + e_C$$

$$\boxed{\begin{aligned} a_C &= \frac{\langle\rho\rangle_C^{n+1}}{\Delta t}|V_C| + \sum_f w_C\left([\rho]\frac{\Delta V_f}{\Delta t} + \rho^{-}\phi_f^{n}\right) \\ a_N &= -\left([\rho]\frac{\Delta V_f}{\Delta t} + \rho^{-}\phi_f^{n}\right)w_N \\ e_C &= \frac{\langle\rho\mathbf{U}\rangle_C^{n}}{\Delta t}|V_C| \end{aligned}}$$

- **Pressure correction:**

$$\boxed{\sum_f \left(\frac{1}{a_C}\right)_f \left(\nabla p_d\cdot\mathbf{S}_f\right)_f^{n+1} = \sum_f \left(\frac{\mathcal{H}\left(\langle\mathbf{U}\rangle_N^{m}\right)}{a_C}\right)_f \cdot\mathbf{S}_f - \left(\frac{1}{a_C}\right)_f \left((\mathbf{g}\cdot\mathbf{x})\nabla\rho\cdot\mathbf{S}_f\right)_f^{n+1}}$$

- Pressure gradient discretized using linear approximation:

$$\left(\nabla p_d\cdot\mathbf{S}_f\right)_f^{n+1} \approx \frac{(p_d)_N^{n+1} - (p_d)_C^{n+1}}{|\mathbf{d}_{CN}|}|\mathbf{S}_f|$$

- Discretization of gravitational force and face gradient :

$$\left((\mathbf{g}\cdot\mathbf{x})\nabla\rho\cdot\mathbf{S}_f\right)_f^{n+1} \approx (\mathbf{g}\cdot\mathbf{x}_f)\left(\nabla\rho\cdot\mathbf{S}_f\right)_f^{n+1} \approx (\mathbf{g}\cdot\mathbf{x}_f)\frac{\rho_N^{n+1} - \rho_C^{n+1}}{|\mathbf{d}_{CN}|}|\mathbf{S}_f|$$

Incompressible Multiphase Flow Equations (interIsoFoam.C)

interIsoFoam: Solver for two incompressible, isothermal immiscible fluids using isoAdvector phase-fraction-based interface capturing. With optional mesh motion and mesh topology changes, including adaptive re-meshing.

Algorithm of the interIsoFoam solver [1] :

1.) Initialize fields: U, p_d and ρ

2.) while Time loop do

3.) Compute Δt according to Courant number

4.) while Outer loop do

5.) Reconstruct the interface (Compute interface center and normal)

6.) Advect interface (Update ρ)

7.) while Inner loop do

8.) Build pressure equation (with updated velocity)

9.) Solve pressure equation (Update p_d)

10.) Update velocity using the pressure gradient (Update U)

11.) end while

12.) end while

13.) end while

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Info<< "\nStarting time loop\n" << endl;

while (runTime.run())
{
    #include "readDyMControls.H"
    #include "CourantNo.H"
    #include "alphaCourantNo.H"
    #include "setDeltaT.H"

    ++runTime;

    Info<< "Time = " << runTime.timeName() << nl << endl;

    // --- Pressure-velocity PIMPLE corrector loop
    while (pimple.loop())
    {
        if (pimple.firstIter() || moveMeshOuterCorrectors)
        {
            mesh.update();

            if (mesh.changing())
            {

                gh = (g & mesh.C()) - ghRef;
                ghf = (g & mesh.Cf()) - ghRef;

                MRF.update();

                if (correctPhi)
                {
                    // Calculate absolute flux
                    // from the mapped surface velocity
                    phi = mesh.Sf() & Uf();

                    #include "correctPhi.H"

                    // Make the flux relative to the mesh motion
                    fvc::makeRelative(phi, U);

                    mixture.correct();
                }
```

Time loop starts

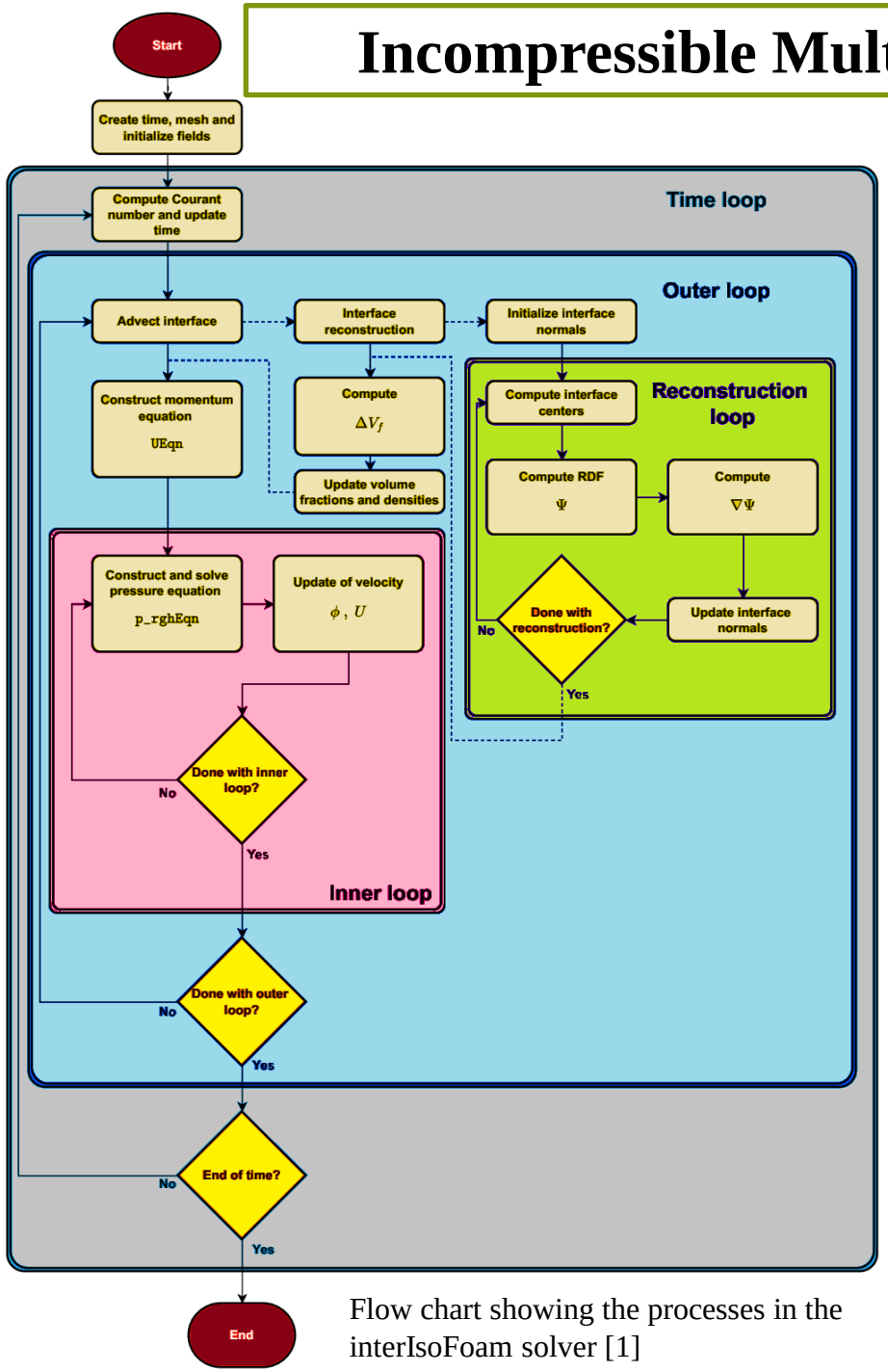
Settings for dynamic mesh controls, Courant number calculations, alpha Courant number calculations, and setting the time step.

Start of pimple outer loop

Updates the mesh to account for any mesh motion

if block checks if it's the first iteration or if mesh motion correction is needed and updates geometric quantities (gravitational terms, mesh velocity), rotating frame effects (MRF)

Incompressible Multiphase Flow Equations (interIsoFoam.C)



Flow chart showing the processes in the interIsoFoam solver [1]

```
140                 if (checkMeshCourantNo)
141                 {
142                     #include "meshCourantNo.H"
143                 }
144             }
145         }
146
147         #include "alphaControls.H"
148         #include "alphaEqnSubCycle.H"
149
150         mixture.correct();
151
152         if (pimple.frozenFlow())
153         {
154             continue;
155         }
156
157         #include "UEqn.H"
158
159         // --- Pressure corrector loop
160         while (pimple.correct())
161         {
162             #include "pEqn.H"
163         }
164
165         if (pimple.turbCorr())
166         {
167             turbulence->correct();
168         }
169     }
170
171     runTime.write();
172
173     runTime.printExecutionTime(Info);
174 }
175
176 Info<< "End\n" << endl;
177
178 return 0;
179 }
180
181
182 // ************************************************************************* //
183
```

alphaEqnSubCycle.H, alpha is calculated using the specified number of sub-cycles mentioned in alphaControls.H. The calculation is done in the file **alphaEqn.H**

Momentum Equation Solver

Pressure corrector loop within the PIMPLE loop (iterates until convergence is achieved for the pressure field).

Incompressible Multiphase Flow Equations (interIsoFoam solver)

alphaEqn.H

```
{
    // Temporarily making U relative to mesh motion
    if (mesh.moving())
    {
        U -= fvc::reconstruct(mesh.phi());
    }

    // Updating alpha1
    advector.advect();

    // Making U absolute again after advection step
    if (mesh.moving())
    {
        U += fvc::reconstruct(mesh.phi());
    }

    #include "rhofs.H"
    rhoPhi = advector.getRhoPhi(rho1f, rho2f);

    alpha2 = 1.0 - alpha1;
    mixture.correct();
}
```

Member functions of the advector object are called.

Pointing to **getRhoPhi** of the isoAdvection class

isoAdvection.H

```
//- Return mass flux
tmp<surfaceScalarField> getRhoPhi
(
    const dimensionedScalar rho1,
    const dimensionedScalar rho2
) const
{
    return tmp<surfaceScalarField>
    (
        new surfaceScalarField
        (
            "rhoPhi",
            (rho1 - rho2)*dVf_/mesh_.time().deltaT() + rho2*phi_
        )
    );
}
```

RHS is the **getRhoPhi** term

$$\sum_f \mathbf{U}_f \left([\rho]\Delta V_f + \rho^- \phi_f^n \Delta t \right) = \Delta t \sum_f \mathbf{U}_f \left(\frac{[\rho]\Delta V_f}{\Delta t} + \rho^- \phi_f^n \right)$$

Ueqn.H

```
MRF.correctBoundaryVelocity(U);

fvVectorMatrix UEqn
(
    fvm::ddt(rho, U) + fvm::div(rhoPhi, U)
  + MRF.DDt(rho, U)
  + turbulence->divDevRhoReff(rho, U)
 ==
    fvOptions(rho, U)
);

UEqn.relax();

fvOptions.constrain(UEqn);

if (pimple.momentumPredictor())
{
    solve
    (
        UEqn
     ==
        fvc::reconstruct
        (
            (
                mixture.surfaceTensionForce()
              - ghf*fvc::snGrad(rho)
              - fvc::snGrad(p_rgh)
            ) * mesh.magSf()
        )
    );

    fvOptions.correct(U);
}
```

Relaxation to improve the stability of the numerical solution.

To introduce additional source terms or modify existing equations.

If momentum predictor is active, this block solves the modified momentum equation. It includes terms for surface tension force, gravitational effects, and pressure gradient.

- **UEqn** is an object of the fvVectorMatrix class, which is a type definition of fvMatrix<vector> i.e. an object of the fvMatrix class with vector as template parameter.
- LHS of the momentum equation mention the time derivative, **fvm::ddt(rho, U)** and the nonlinear convection term, **fvm:div(rhoPhi, U).**
- **rhoPhi** is an object of the type surfaceScalarField and is computed in the **alphaEqn.H** file.

Incompressible Multiphase Flow Equations (interIsoFoam solver)

pEqn.H (Pressure-velocity coupling)

```
{
    if (correctPhi)
    {
        rAU.ref() = 1.0/UEqn.A();
    }
    else
    {
        rAU = 1.0/UEqn.A();
    }

    surfaceScalarField rAUf("rAUf", fvc::interpolate(rAU()));
    volVectorField HbyA(constrainHbyA(rAU()*UEqn.H(), U, p_rgh));
    surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::flux(HbyA)
      + MRF.zeroFilter(fvc::interpolate(rho*rAU())*fvc::ddtCorr(U, phi, Uf))
    );
    MRF.makeRelative(phiHbyA);
```

Update of a_P from U freshly computed

Interpolation of rAU object on the faces

Update of a_P/H from U freshly computed

Projection of a_P/H over the cell faces

```
    surfaceScalarField phig
    (
        (
            mixture.surfaceTensionForce()
          - ghf*fvc::snGrad(rho)
        )*rAUf*mesh.magSf()
    );

    phiHbyA += phig;

    // Update the pressure BCs to ensure flux consistency
    constrainPressure(p_rgh, U, phiHbyA, rAUf, MRF);

    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvm::laplacian(rAUf, p_rgh) == fvc::div(phiHbyA)
        );

        p_rghEqn.setReference(pRefCell, getRefCellValue(p_rgh, pRefCell));

        p_rghEqn.solve(mesh.solver(p_rgh.select(pimple.finalInnerIter())));

        if (pimple.finalNonOrthogonalIter())
        {
            phi = phiHbyA - p_rghEqn.flux();

            p_rgh.relax();

            U = HbyA + rAU()*fvc::reconstruct((phig - p_rghEqn.flux())/rAUf);
            U.correctBoundaryConditions();
            fvOptions.correct(U);
        }
    }

    #include "continuityErrs.H"

    // Correct Uf if the mesh is moving
    fvc::correctUf(Uf, U, phi);

    // Make the fluxes relative to the mesh motion
    fvc::makeRelative(phi, U);

    p == p_rgh + rho*gh;
```

Pressure equation $\nabla \cdot \left(\frac{1}{a_P}\nabla p\right) = \nabla \cdot \left(\frac{H(U)}{a_P}\right)$ is defined

Pressure equation is solved

Here, we recover the right value of the flux velocity, $\emptyset = \frac{H(U)}{a_P} \cdot S - \frac{1}{a_P}\nabla p \cdot S$

Velocity corrector stage: $U_P = \frac{1}{a_P}H(U) - \frac{1}{a_P}\nabla p$

Evaluate the velocity fields

Print the continuity error on screen

Incompressible Multiphase Flow Equations (interIsoFoam solver)

System/fvSolution

```
solvers
{
    "alpha.water.*"
    {
        isoFaceTol       1e-6;
        surfCellTol      1e-6;
        nAlphaBounds     3;
        snapTol          1e-12;
        clip             true;

        nAlphaSubCycles 1;
        cAlpha           1;
    }

    "pcorr.*"
    {
        solver           PCG;
        preconditioner   DIC;
        tolerance        1e-10;
        relTol           0;
    }

    p_rgh
    {
        solver           GAMG;
        smoother         DICGaussSeidel;
        tolerance        1e-07;
        relTol           0.05;
    }

    p_rghFinal
    {
        $p_rgh;
        tolerance        1e-07;
        relTol           0;
    }

    U
    {
        solver           PBiCGStab;
        preconditioner   DILU;
        tolerance        1e-06;
        relTol           0;
    }
}
```

alpha controls for isoAdvector solution

Linear-solver to solve for pressure flux correction for a dynamic simulation

GAMG solver (generalized geometric-algebraic multigrid solver) for solving the pressure equation.

Linear-solver to solve for velocity

PIMPLE algorithm controls

```
PIMPLE
{
    momentumPredictor no;
    nCorrectors      3;
    nOuterCorrectors 1;
    nNonOrthogonalCorrectors 0;
    pRefCell         0;
    pRefValue        0;
}
```

System/controlDict

```
application     interIsoFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         5;

deltaT          0.001;

writeControl    adjustable;

writeInterval   0.02;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           10;
maxAlphaCo      0.5;

maxDeltaT       1;
```

Select Solver name

Initial time

Final time

Time-step or Delta T

Writing interval

Writing format

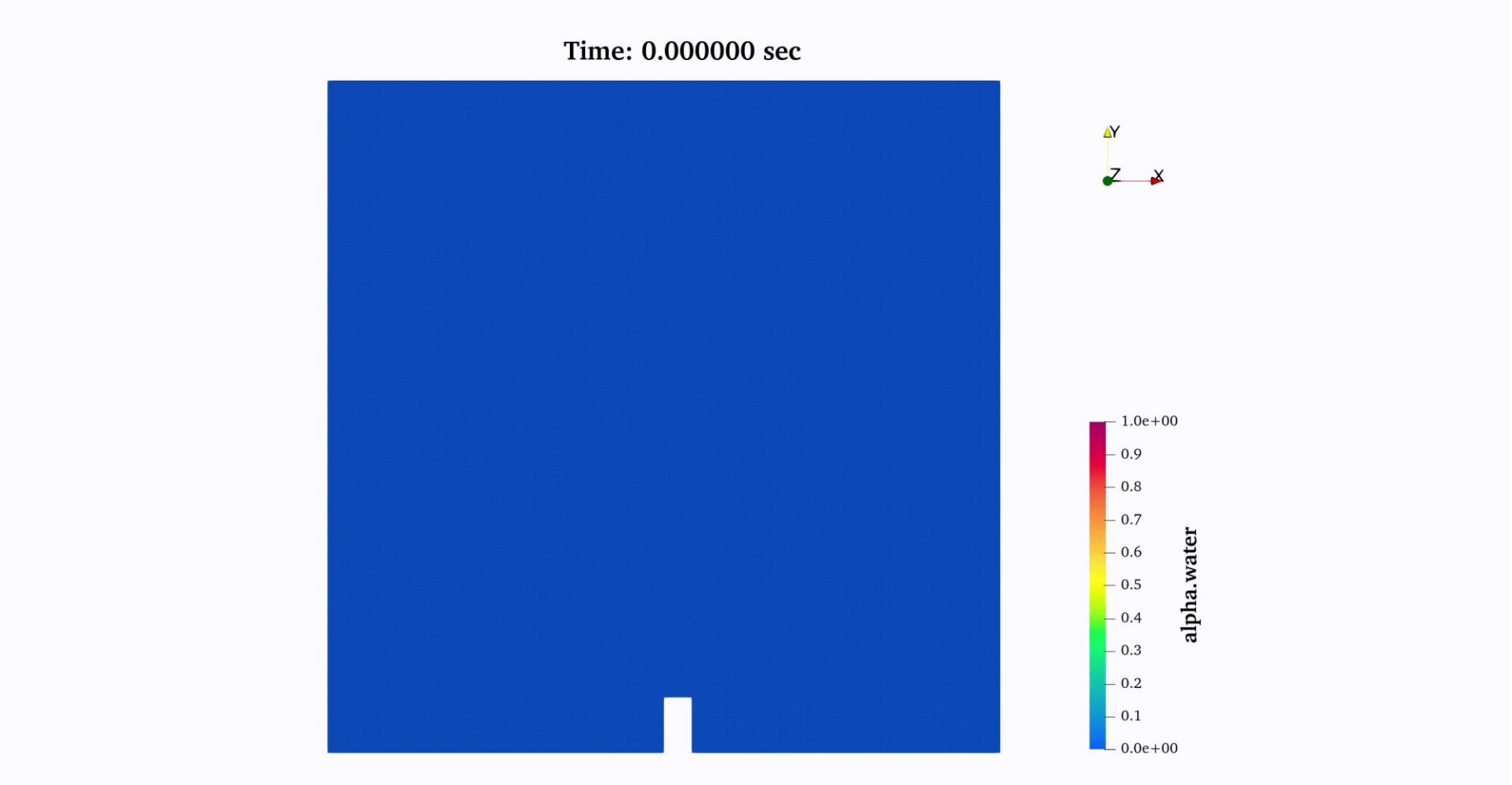
Writing precision

Time precision

Courant no.s (Co and alphaCo)

Sample tutorial using interIsoFoam solver: damBreak case (2D)

Problem Set - III




danke
謝謝
ngiyabonga
teşekkür ederim
спасибо
dank je
thank you
gracias
bedankt
mochchakkeram
dziękuje
go raibh maith agat
obrigado
sukriya
kop khun krap
arigato
dakujem
grazie
terima kasih
merci
감사합니다
мерси


Q&A
How
Who
What
When
Where
Why