CFD on HPC – OpenFOAM

03 – Advanced usage

Aleksander GRM – June, 2025

Content

Introduction

- **Templates**: use templates to start case
  `$> cp $FOAM_ETC/templates/... case_dir`
- **Dictionaries**: use `foamGet` to copy templated dictionary files
  `$> foamGet forcesIncompressible`
- **Information**: use `foamInfo` to get help on foam item
  `$> foamInfo incompressibleFluid`
- **Dictionary manipulation**: use `foamDictionary` to edit dictionary files
  `$> foamDictionary [OPTIONS] <dictionary file>`

Nice example in folder

- `day_02/02_test_case-cavity_with_hole/snappyHexMesh/steadyState/` look into `readme.txt` file

Example

```
Fancy corrections of constant/polyMesh/boundary file symmetryPlane -> empty

- foamDictionary constant/polyMesh/boundary -keywords
- foamDictionary constant/polyMesh/boundary -entry entry0 -keywords

Front:
- foamDictionary constant/polyMesh/boundary -entry entry0/front -keywords
- foamDictionary constant/polyMesh/boundary -entry entry0/front/type -value
- foamDictionary constant/polyMesh/boundary -entry entry0/front/type -set empty

here are two choices, both works
- foamDictionary constant/polyMesh/boundary -entry entry0/front/inGroups -set "List<word> 1(empty)"
- foamDictionary constant/polyMesh/boundary -entry entry0/front/inGroups -remove
```

Hydrofoil case

- describe basic case philosophy
- describe two different meshing approaches
  - OpenFOAM foil `blockMesh`: change angle of attack with velocity vector rotation
  - GMSH foil `mesh`: change angle of attack with geometry rotation
- Describe two different solutions (OF module `foamRun`)
  - steady state – `SIMPLE` algorithm
  - transient – `PIMPLE` algorithm
- show mesh generation for different angles of attack (compare with fixed mesh),
- solution initialization with `potentialFoam` integrated in `foamRun`,
- compare solutions of the case for angle of attack 20° (turbulent and laminar solver).

Simple mixer case

Problem: mixing `dye` with `paint`

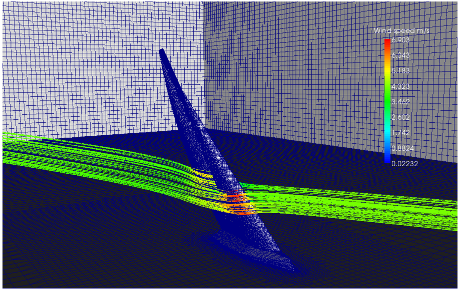
- steady state – `SIMPLE` algorithm – show run without the dye, only paint flow
- transient – `PIMPLE` algorithm – run complete case with dye inflow until 300s

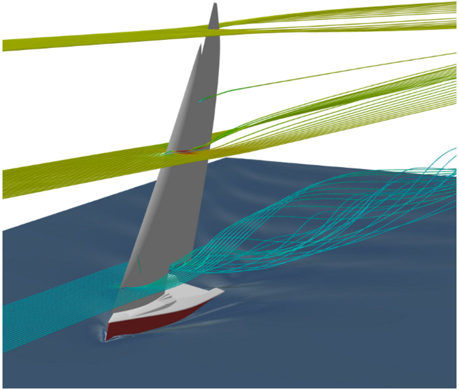
Meshing procedure

- `blockMesh`: background mesh procedure
- `snappyHexMesh`: steps to generate computational mesh
  - `baffles` objects in OpenFOAM mesh (2D object in 3D space)
  - `surfaceFeatures` detect featured edges
  - set `refinement` levels
  - use `snappyHexMeshConfig` to generate automatically dictionary files
  - edit all generated dictionary files

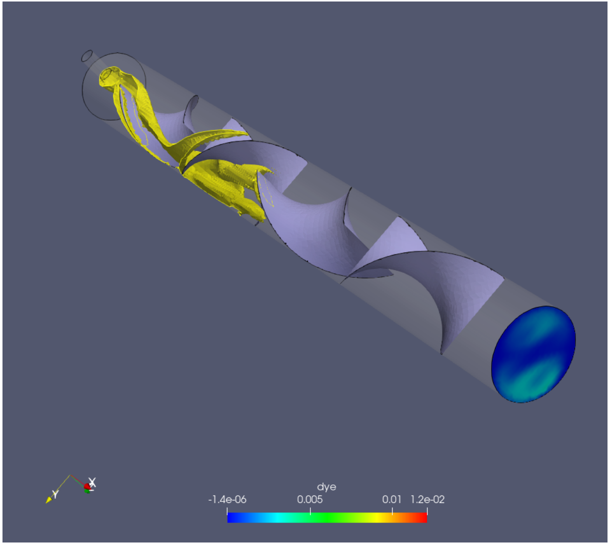
OpenFOAM baffle mesh element

Typical is simulation of sail performance. It is a 2D object in 3D space



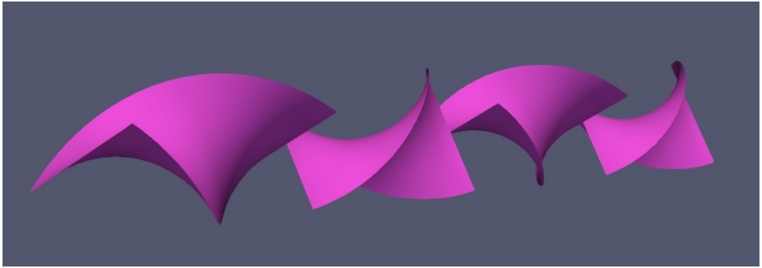


Static mixer case



For 2D geometry embedded into 3D space in OpenFOAM snappyHexMesh use **baffle** element type.

In mixer baffles are static curved surfaces installed in pipe to mix dye with paint.



Generate **baffles** with snappyHexMesh

- `snappyHexMeshConfig -help`
- Set baffles from file `constant/geometry/baffles.obj`
  `$> snappyHexMeshConfig -baffles '(baffles)'`

Surface geometry files are automagically recognised; look at the output

```
+ pipe
+ File: "pipe.obj"
+ Bounding box: (0 -0.05 -0.09) (1 0.05 0.05)
+ Closed surface
+ External boundary surface
+ Inlet regions: inletMain, inletSide
+ Outlet regions: outlet

+ baffles
+ File: "baffles.obj"
+ Bounding box: (0.1 -0.05 -0.05) (0.9 0.05 0.05)
+ Open surface
+ Baffle wall surface
```

`snappyHexMeshConfig` generates 3 system mesh files

- `blockMeshDict`
- `surfaceFeaturesDict`
- `snappyHexMeshDict`

blockMeshDict

geometry file `pipe.obj` forms an external boundary so,
it is possible to clear the boundary

```
$> snappyHexMeshConfig -baffles '(baffles)' -clearBoundary
```

Comment on blockMeshDict file

Generate `baffles` with snappyHexMesh

surfaceFeaturesDict file containing surface feature capturing information
generates additional geometry files with extension `*.eMesh`

```
$> snappyHexMeshConfig -baffles '(baffles)'
     -clearBoundary -explicitFeatureSnap
```

- `implicitFeatureSnap`: automatically identifies surface features
  - may not pickup specific features, like ends, edges, ...
- `explicitFeatureSnap`: features defined in a file
  - user must specify a file
  - complete control over features identification and refinement

Surface Features dictionary


```
surfaces                    <-- list of surface geometry files
(
    "pipe.obj"
    "baffles.obj"
);

includedAngle   150;        <-- feature identification parmeter

subsetFeatures
{
    nonManifoldEdges yes;    <-- edges connected to more than 2 edges
    openEdges        yes;    <-- edges connected to 1 face
}

trimFeatures

{
    minElem         0;       <-- minimum number of edges in feature
    minLen          0;       <-- minimum length of feature
}
writeObj        yes;        <-- write additional obj files to visualise features
```

Generate computational mesh

Generate all mesh dictionaries

```
snappyHexMeshConfig \
  -baffles '(baffles)' \
  -clearBoundary \
  -explicitFeatures \
  -refinementLevel 1 \.            <-- default level
  -surfaceLevels '((pipe 2))' \      <-- pipe refinement level
  -refinementRegions '((pipe 1))' \  <-- internal region
  -nCellsBetweenLevels 1           <-- cells between refinement levels
```

To create the mesh, must run

- `blockMesh`
- `surfaceFeatures`
- `snappyHexMesh -overwrite` (overwrite creates polyMesh)

Transport properties are set in dictionaries

- `constant/physicalProperties`:
  transport material properties: viscosity, density, ...
- `constant/momentumTransport`:
  fluid flow momentum transport properties: flow type, turbulence model, viscosity model, ...
- `constant/thermophysicalTransport`:
  energy transport properties: specific conduction, enthalpy, ...

Paint Transport Model

Paint can be described as **generalized newtonian fluid**, viscosity is not constant, but strain rate $\dot{\gamma}$ dependant

$$\nu = f(\dot{\gamma}), \quad \dot{\gamma} = \sqrt{2}|\boldsymbol{D}|, \quad \boldsymbol{D} = \mathrm{sym}\,(\nabla \boldsymbol{v})$$

where: $\dot{\gamma}$ is strain rate, $\boldsymbol{D}$ is symmetric part of velocity gradient

Shear stress

$$\boldsymbol{\tau} = 2\rho\nu\boldsymbol{D}$$

Listing models

```
$> foamToC -table generalisedNewtonianViscosityModel
$> foamInfo BirdCarreau
```

Surface Features dictionary

```
Contents of table generalisedNewtonianViscosityModel:
    BirdCarreau                                  libmomentumTransportModels.so
    Casson                                       libmomentumTransportModels.so
    CrossPowerLaw                                libmomentumTransportModels.so
    HerschelBulkley                              libmomentumTransportModels.so
    Newtonian                                    libmomentumTransportModels.so
    powerLaw                                     libmomentumTransportModels.so
    strainRateFunction                           libmomentumTransportModels.so
```

Bird-Carreau viscosity model

For **paint** use the **Bird-Carreau** strain rate $\dot{\gamma}$ dependant viscosity model

$$\nu = \nu_\infty + (\nu_0 - \nu_\infty)[1 + (k\,\dot{\gamma})^a]^{(n-1)/a}$$

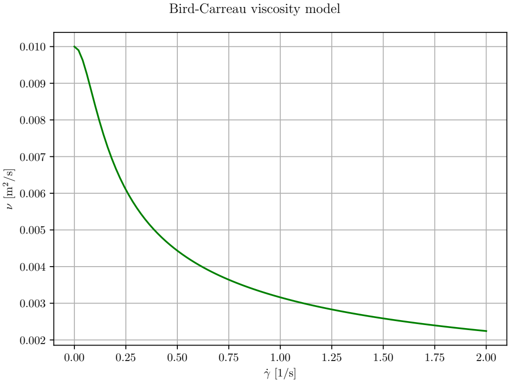
$$\nu_0 = 0.01\,\mathrm{m}^2/\mathrm{s}$$

$$\nu_\infty = 0.000\,01\,\mathrm{m}^2/\mathrm{s}$$

$$k = 10$$

$$n = 0.5$$

$$a = 2 \quad \text{(by default)}$$



Transport model

Choice of two transport models

- **laminar**: slow fluid flow, minimal vorticity
- **turbulent**: when not laminar

All transport models are set in the `constant/momentumTransport` file.

Laminar transport model


```
simulationType  laminar;

laminar
{
  model          generalizedNewtonian;

  viscosityModel  BirdCarreau;
  nuInf           1e-5;
  k               10;
  n               0.5
}
```

```
simulationType RAS;

RAS
{
  model           kOmegaSST;
  turbulence        on;
  printCoeffs       on;

  viscosityModel  BirdCarreau;
  nuInf             1e-5;
  k                 10;
  n                 0.5
}
```

Simulate flow of paint only. Test laminar and turbulent transport model.

Transport model

- **laminar**: iterate until 250 steps
- **turbulent**: iterate until 250 steps

Inlet velocity

- **main**: $1.0\,\mathrm{m/s}$
- **side**: $0.5\,\mathrm{m/s}$ (dye inflow is zero!)

Comment on constant and system files!

Include dye flow at side inlet. Test laminar and turbulent transport model.

Transport model

- **laminar**: iterate until 2.0 s
- **turbulent**: iterate until 2.0 s

Scalar inlet – Dye source

- **main**: 0.0
- **side**: 1.0

Dye (scalar field) dimension is arbitrary. Normally dimension less.

Comment on constant and system files!

- for transient simulation **PIMPLE** solver is used
- for steady-state simulation **SIMPLE** solver is used

Courant number or **CFL** number

- **definition**:

$$\mathrm{CFL} = \frac{U_x\,\Delta t}{\Delta x}$$

  - U_x: velocity in cell
  - Δt: time step
  - Δx: cell size in flow direction

- **explicite** solver: $\mathrm{CFL} < 1$
- **implicite** solver: CFL is more an accuracy consideration

Use some rough approximations in the estimation

- use `checkMesh` to find cell size Δx

  `Min volume = 1.7680364e-09` $\Delta x = \sqrt[3]{2e-9} \approx 1.3\,\mathrm{mm}$

- steady state max velocity $\approx 2.0\,\mathrm{m/s}$

- estimated initial **time step**:

$$\Delta t = \mathrm{CFL}\frac{\Delta x}{U_x} = 1.0\frac{1.3\,\mathrm{mm}}{2.0\,\mathrm{m/s}} \approx 0.6\,\mathrm{ms}$$

CFL condition

Try to play with `maxCo` (max CFL) in `system/controlDict`.

Warning: change in CFL is related to simulation accuracy.
Test dye travel distance for concentration 0.05!

Run simulation with different CFL and measure time of simulation

- use `maxCo` $= 1.0$
- use `maxCo` $= 2.0$
- use `maxCo` $= 4.0$

Use `foamLog` utility to extract simulation step data and plot using `GnuPlot`!

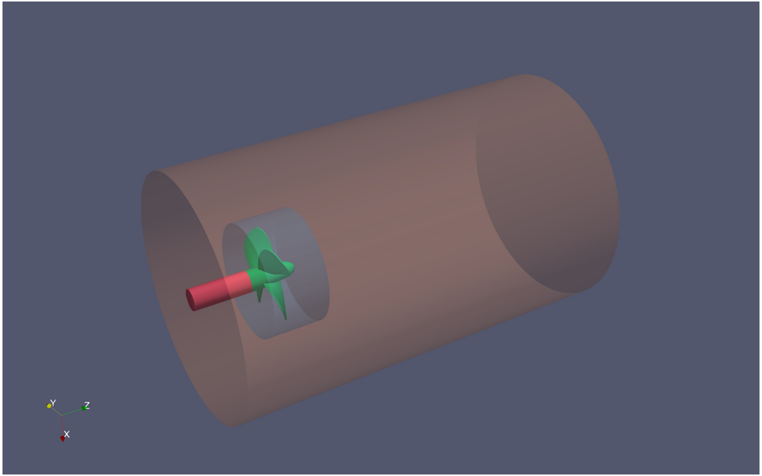
```
File: system/controlDict

adjustTimeStep  yes; // time adaptive scheme ON (use CFL limit)
maxCo           1;   // max CFL number {1,2,4}

Follow:
 - time of execution
 - number of iterations in U and p
```

Ship resistance case

Case: Calculation of ship resistance force

- describe basic case philosophy
- describe meshing approach
  - `blockMesh`: anisotropic background mesh
  - `refineMesh`: different refinement mesh regions
  - `snappyHexMesh`: addition of boundary layer
  - `additional feature`: mesh refinement in bow and stern region (TODO)
- compute quasi steady state
  - steady state – `PIMPLE` algorithm with `localEuler` – pseudo transient
  - `setFields`: set air and water region
  - `incompressibleVoF`: solver module for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach

Propeller – Open water test case



case zones: external, rotating, shaft, propeller

MRF approach.

- **velocity**
  velocity is composed of two terms: **relative** and **rotating**

$$\boldsymbol{v} = \boldsymbol{v}_{\text{rel}} + \omega \times \boldsymbol{r}$$

$\boldsymbol{v}_{\text{rel}}$ - relative velocity (inlet flow)
ω - angular velocity (propeller rotation)
$\boldsymbol{r}$ – radius vector in rotating zone

- **momentum** convection
  use of a relative velocity for convective flux

$$\left(\boldsymbol{v} \cdot \nabla\right)\boldsymbol{v} = \left(\boldsymbol{v}_{\text{rel}} \cdot \nabla\right)\boldsymbol{v} + \omega \times \boldsymbol{v}$$

Case: Calculation of ship propeller thrust/moment.

- describe basic case philosophy
- describe meshing approach
  - `blockMesh`: radially symmetric background mesh
  - `rotatingZone`: define rotating mesh regions
  - `snappyHexMesh`: rotating zone
  - `additional feature`: mesh refinement only in blade region and boundary layer (TODO)
- compute steady state
  - steady state – `PIMPLE` algorithm with `MRF` source
  - `movingWallVelocity`: set as velocity BC at rotating surface (propeller) to obtain rotationžđ
  - `incompressibleVoF`: solver module for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach

Sponsorship

EURO
SLING

Thank you for attention!

EuroHPC Joint Undertaking

REPUBLIC OF SLOVENIA
MINISTRY OF HIGHER EDUCATION, SCIENCE AND INNOVATION

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101101903. The JU receives support from the Digital Europe Programme and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia.