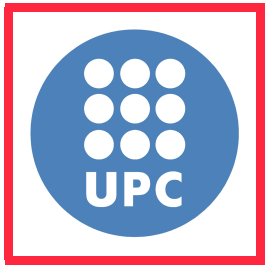


OpenFOAM workshop for beginners: Hands-on training

Jibran Haider [a, b]

Erasmus Mundus PhD (SEED) candidate in computational mechanics

[a] Zienkiewicz Centre for Computational Engineering (ZCCE),
College of Engineering, Swansea University, UK

[b] Laboratori de Càlcul Numèric (LaCàN),
Universitat Politècnica de Catalunya (UPC), BarcelonaTech, Spain

Session 1: *31st May 2016* (09:00 - 12:00)
Session 2: *3rd June 2016* (10:00 - 13:00)

Scheme of presentation

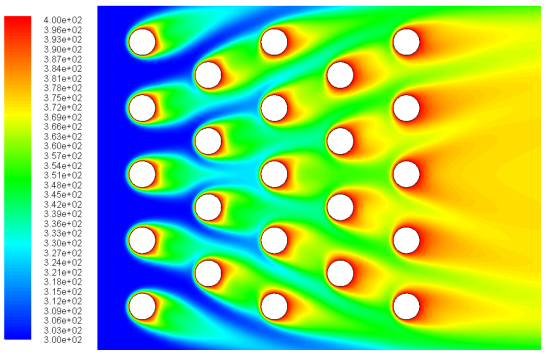
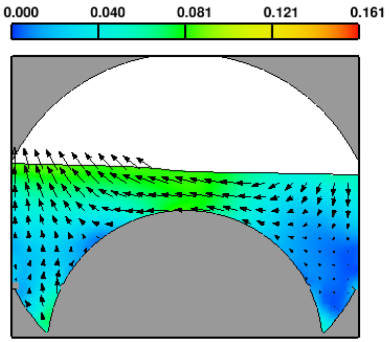
Scheme of presentation

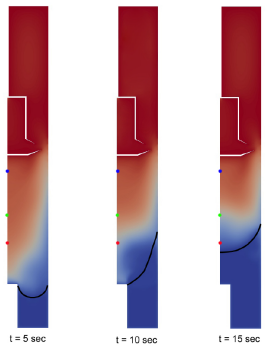
Contents of this section

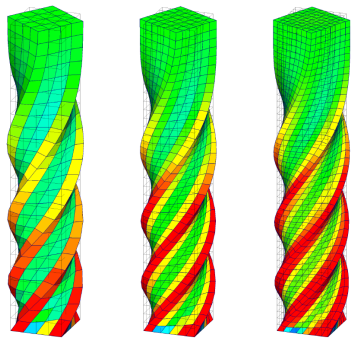
We will look at the following:

Experience in computational mechanics

Year	Topic	Software	Institution
[2009]	CFD and experimental study of flow over cylindrical fins.	FLUENT	NUST
[2012]	Numerical simulation of sloshing phenomena in complex tanks.	FLOW-3D	Universität Stuttgart & DLR
[2013]	Modeling of phase change phenomena through a liquid-vapor interface in microgravity.	OpenFOAM	Universität Stuttgart & DLR
[2014]	A first order hyperbolic framework for large strain computational solid dynamics.	OpenFOAM	Swansea University & UPC





Introduction

Please login to your workstations.

Non-LaCaN members:

`Username` : `course*` $\quad * = 1, 2, 3.....$

`Passwords` : `OpenFOAM2016`

Open the terminal (CTRL + ALT + T).

Terminal is a program that opens a window that lets you interact with the shell.

Course material is located at /lordvader/courses/OpenFoam/.

Open a copy of this presentation:

```
<< evince /lordvader/courses/OpenFoam/presentation/presentation.pdf &
```

Introduction

Objectives of this course:

1. Introduce the basics of **OpenFOAM**.
2. Signify the advantages of using **OpenFOAM**.
3. Utilise the power of linux by extensive use of shell commands.
4. By the end of the course, candidates should be able to run **OpenFOAM** simulations.

Desirable traits

- Basic level of programming in **C++**.
- Familiarization with Linux interface.
- Fundamentals of Computational Fluid Dynamics (**CFD**).

During the workshop we will be using:

- **OpenFOAM** (Version 2.4.0).
- **ParaView** (Version 4.1.0).

Disclaimer

"This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the **OpenFOAM** software and owner of the OPENFOAM® and OpenCFD® trade marks."

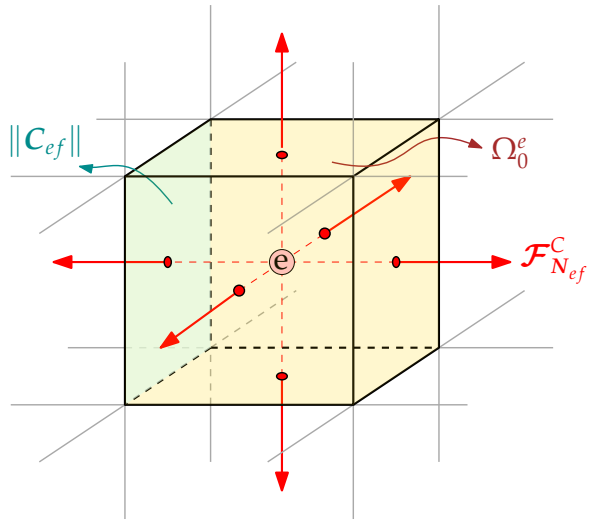
What is OpenFOAM?

Summary

- An **open source** software package
- Developed by **OpenCFD Ltd** at **ESI Group**
- Released under the **GNU General Public License** [link]
- Abbreviation of **O**pen **F**ield **O**peration **A**nd **M**anipulation
- An efficient library of **C++** modules
- Based on the **cell centered Finite Volume Method**
- Extensive **CFD** and **Multi-physics** capabilities
- Under active development [**OpenFOAM** Extend Project]

OpenFOAM

esi get it right®



What is OpenFOAM?

OpenFOAM is a **C++** library of highest quality of programming for solving computational continuum mechanics problems utilising all the features of Objected Oriented Programming (**OOP**).

Key features of OOP

- **Abstraction**
- **Inheritance**
- **Polymorphism**
  Objects of different classes respond differently to functions of the same name.
  Also includes operator overloading.

Template metaprogramming

- **C++** provides template classes.
- General features of the template class are passed on to any other class created from it.
- Therefore it reduces code duplication.

Why OpenFOAM?

		OpenFOAM	Commercial Softwares (ANSYS, ABAQUS etc.)
1.	Cost	✓	✗
2.	Parallel computing	✓	✓
3.	Source code	✓	✗
4.	Redistribution of code	✓	✗
5.	Collaborative development	✓	✗
6.	Documentation	✗	✓
7.	GUI & user friendliness	✗	✓

- ✓ Motivates **collaborative** and **project-driven** development.
- ✓ According to the **GNU GPL v3**, **OpenFOAM** is **free** to download, install, use, modify and distribute.

Development of OpenFOAM

1989	Project was initially started under the title of **FOAM** at Imperial college, London. The idea was to create a simulation platform more powerful than **FORTRAN**. Due to its object oriented features **C++** was chosen as the programming language.
10th Dec 2004	OpenCFD Ltd was founded and first version (v 1.0) of **OpenFOAM** was released under the **GNU GPL** license.
8th Aug 2011	OpenCFD was acquired by Silicon Graphics International (**SGI**).
15th Aug 2012	OpenCFD became a wholly owned subsidiary of ESI group.
15th Dec 2015	**OpenFOAM** v3.0.1 was released.

- Henry Weller and Hrvoje Jasak are the main contributors to its development.
- OPENFOAM® is a registered trade mark of OpenCFD Limited, producer and distributor of the **OpenFOAM** software.

Getting started with OpenFOAM

Installation:

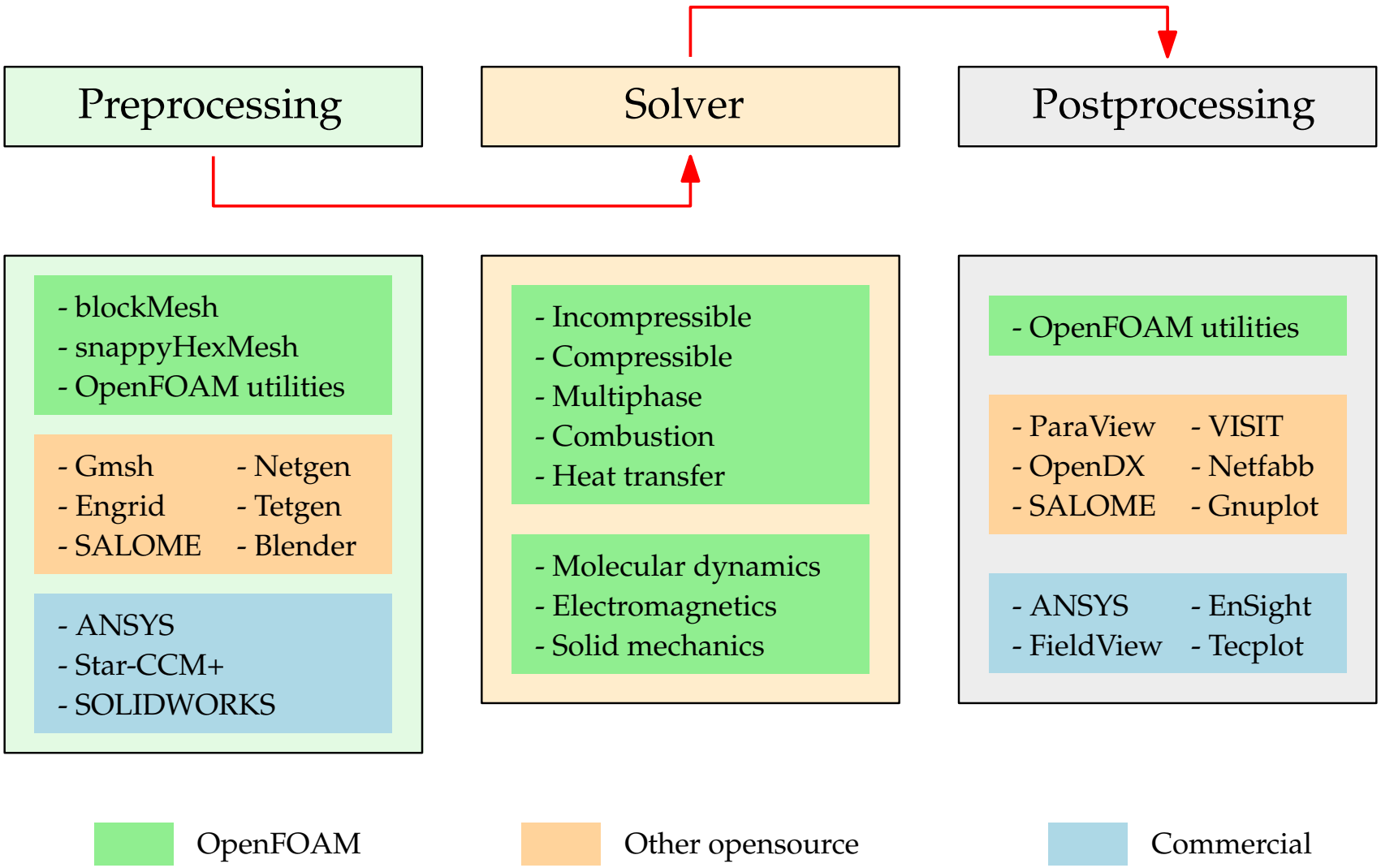
- **OpenFOAM** website [http://www.openfoam.org/download/]
- **OpenFOAM** wiki [https://openfoamwiki.net/index.php/Installation]

References:

- **OpenFOAM** user guide [http://www.openfoam.org/docs/]
- **OpenFOAM** programmer's guide [http://www.openfoam.org/docs/]
- Source code [http://www.openfoam.org/docs/cpp/]
- Book [The OpenFOAM technology primer]
- Research article
  [A tensorial approach to computational continuum mechanics using object-oriented techniques]

✓ Plenty of **free** material available on the web (tutorials, reports, thesis etc).

Simulation workflow



Conventions used for this presentation

OpenFOAM terminologies:

OpenFOAM_terminology

OpenFOAM command for applications:

OpenFOAM_application

Commands on terminal:

`<< command_name`

OpenFOAM environment

Load **OpenFOAM** environment every time you use it.

Load OpenFOAM module:

```
<< source /opt/openfoam240/etc/bashrc
```

Load pyFoam library:

```
<< export PYTHONPATH=/opt/python/lib/python2.7/site-packages:$PYTHONPATH
<< export PATH=/opt/python/bin:$PATH
```

Alternatively for regular **OpenFOAM** use, edit your bashrc file.

Open bashrc file:

```
<< gedit ~/.bashrc
```

Add the following at the end of the file:

```
source /opt/openfoam240/etc/bashrc
export PYTHONPATH=/opt/python/lib/python2.7/site-packages:$PYTHONPATH
export PATH=/opt/python/bin:$PATH
```

Environment variables & aliases

Environment variables are preconfigured to important **OpenFOAM** directories.

Environment variables beginning with FOAM_:

```
<< env | grep ^FOAM_
```

Aliases make use of environment variables to navigate inside **OpenFOAM** sources.

Definition of aliases:

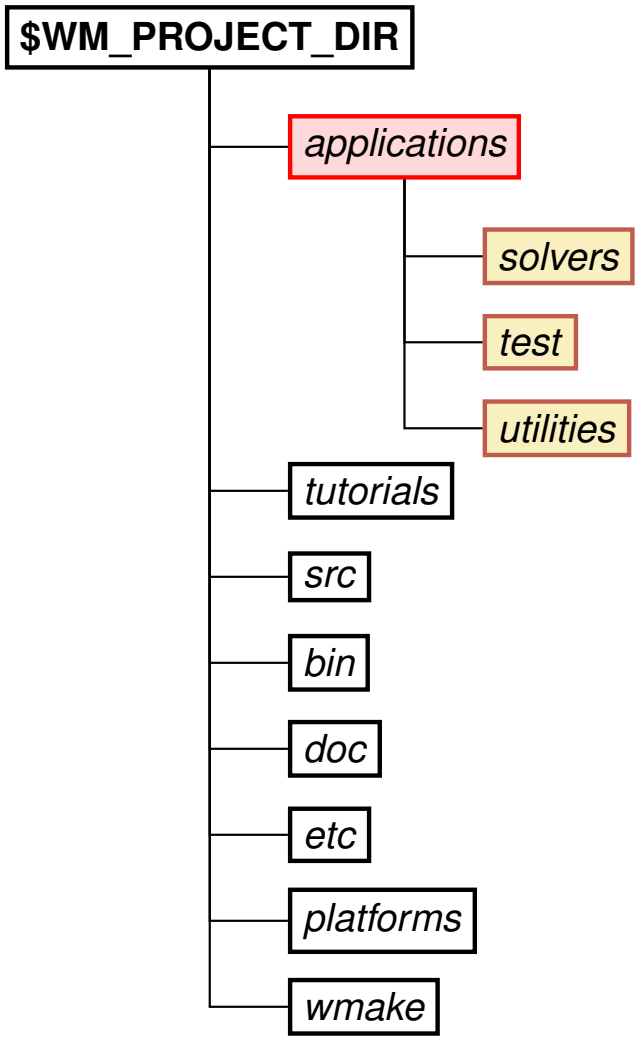
```
<< less -N $WM_PROJECT_DIR/etc/config/aliases.sh
```

Scheme of presentation

Contents of this section

We will look at the following:

Installation directory



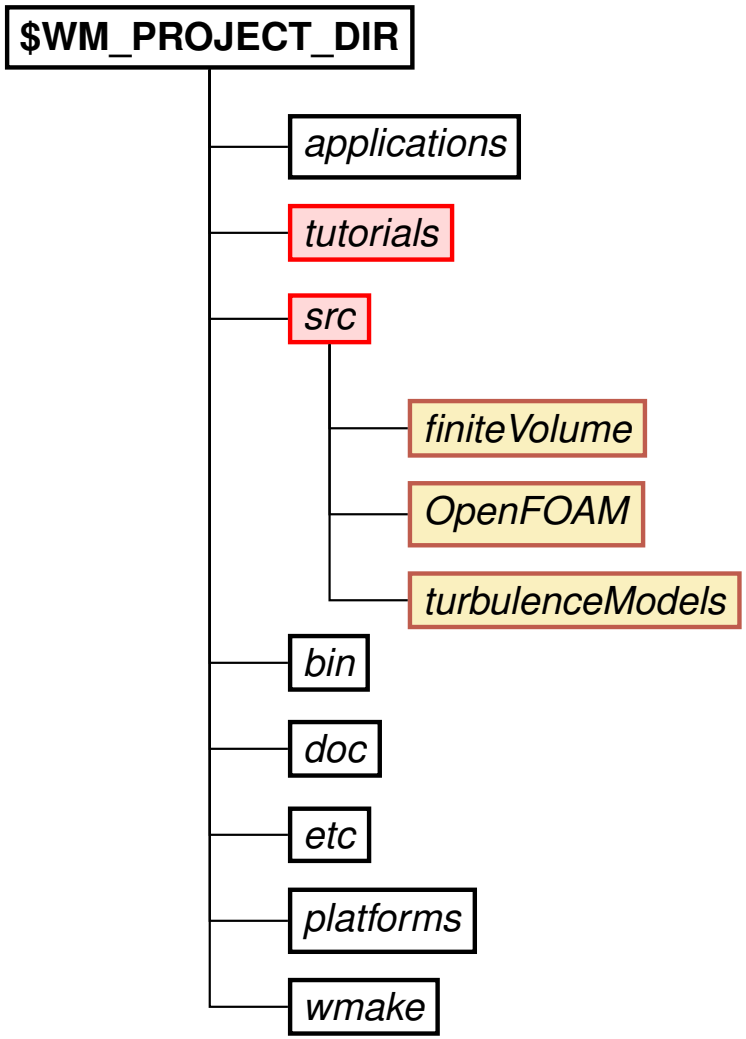
1. *applications*:
   alias: *app* = 'cd $FOAM_APP'.
   This directory contains the source files of all the executables created using the **C++** libraries. It contains the following directories:

   1.1 *solvers*:
   alias: *sol* = 'cd $FOAM_SOLVERS'.
   Source code to solve a particular continuum mechanics problem.

   1.2 *test*:
   Sample codes to help understand the usage of OpenFOAM libraries.

   1.3 *Utilities*:
   alias: *util* = 'cd $FOAM_UTILITIES'.
   Source code to perform pre- and post-processing tasks involving data manipulation and algebraic manipulations.

Installation directory



2. *tutorials*:
   alias: *tut* = 'cd $FOAM_TUTORIALS'.
   Contains tutorials that demonstrate the usage of all solvers and most of the utilities.

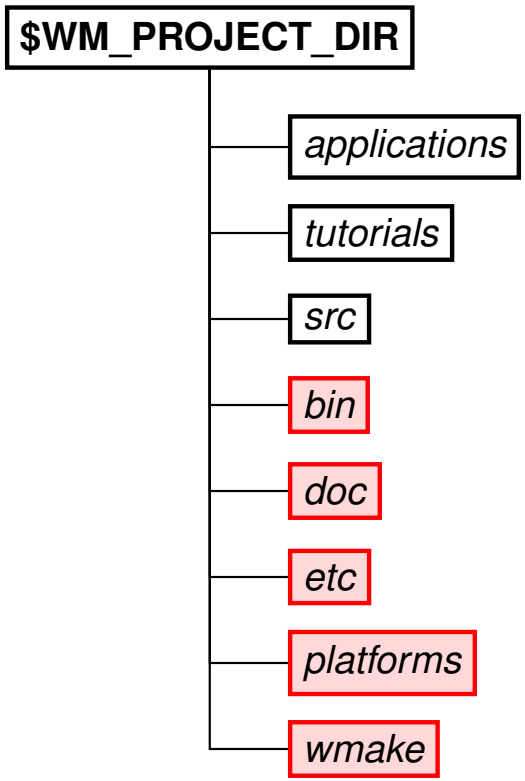
3. *src*:
   alias: *src* = 'cd $FOAM_SRC'.
   It contains several subdirectories which include the source code for all libraries. The important folders are:

   3.1 *finiteVolume*:
   alias: *foamfv* = 'cd $FOAM_SRC/finiteVolume'.
   Includes classes for finite volume space/time discretisation, boundary conditions etc.

   3.2 *OpenFOAM*:
   This core library includes important definitions.

   3.3 *turbulenceModels*:
   Contains libraries for turbulence models.

Installation directory



4. *bin*:
   This directory contains shell scripts such as *paraFoam*, *foamLog* etc.
5. *doc*:
   It contains all the documentation relevant to the version of **OpenFOAM** including:
   - 5.1 User and Programmer's guides
   - 5.2 Doxygen generated documentation
   - 5.3 **OpenFOAM** coding style guide
6. *etc*:
   It contains global **OpenFOAM** dictionaries and setup files.
7. *platforms*:
   The binaries generated during the compilation of the applications and the dynamic libraries are stored here.
8. *wmake*:
   Compiler settings are included in this directory including optimisation flags. It also contains *wmake*, a special make command which understands the **OpenFOAM** file structure.

User's working directory

The path to user's working directory is stored in $WM_PROJECT_USER_DIR.

Check the path of working directory:

```
<< echo $WM_PROJECT_USER_DIR
```

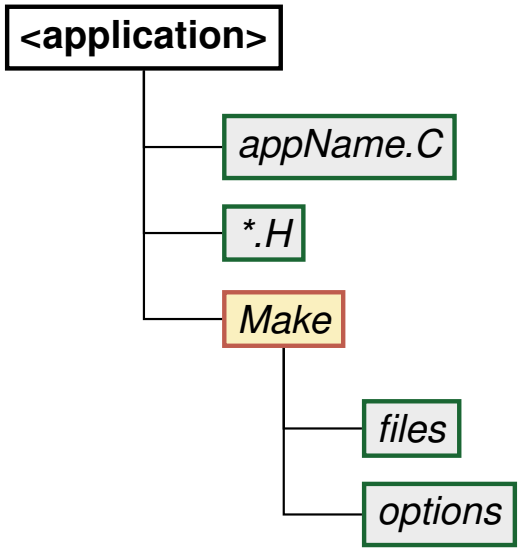
Create the run directory:

```
<< mkdir -p $FOAM_RUN
```

Check the path of 'run' directory:

```
<< echo $FOAM_RUN
    /lordvader/<username>/OpenFOAM/<username>-2.4.0/run
```

Directory structure



1. *appName.C*:
   Contains the main source code of the application.

2. **.H*:
   The necessary header files needed to compile the application. The solvers include *createFields.H* file which declares and initialises the field variables.

3. *Make*:
   This directory contains the compilation instructions and includes the following files:

   3.1 *files*:
   List of source files (*.C) needed to compile the application and the executable name.

   3.2 *options*:
   Specifies the directories to search for the included header files (*.H) and libraries needed to link the application against.

Standard solvers

✓ Over 70+ **OpenFOAM** solvers.

Basic CFD solvers:

	Solver	Description
1.	*laplacianFoam*	Solves a Laplace equation.
2.	*potentialFoam*	Solves for potential flow.

Incompressible flow:

	Solver	Description
1.	*icoFoam*	Transient solver for incompressible, laminar flow of Newtonian fluids.
2.	*simpleFoam*	Steady state solver for incompressible, turbulent flow.
3.	*pisoFoam*	Transient solver for incompressible flow.

Standard solvers

Compressible flow:

	Solver	Description
1.	*rhoCentralFoam*	Density based compressible flow solver based on central upwind scheme.
2.	*sonicFoam*	Transient solver for trans-sonic/supersonic, laminar/turbulent flow of a compressible gas.

Muliphase flow:

	Solver	Description
1.	*interFoam*	Solver for 2 incompressible, isothermal and immiscible fluids based on the **VOF** method.
2.	*interPhaseChangeFoam*	Solver for 2 incompressible, isothermal and immiscible fluids with a phase change model.
3.	*multiPhaseEulerFoam*	Solver for multiple compressible fluid phases with heat transfer.

Standard solvers

Other solvers:

	Solver	Description
1.	*solidDisplacementFoam*	Transient solver for linear-elastic, small-strain solver deformation of a solid body.
2.	*mdFoam*	Molecular dynamics solver for fluids.
3.	*buoyantSimpleFoam*	Steady state solver for buoyant, turbulent flow of compressible fluids.

Solver name describe their functionality:

simple: **SIMPLE** algorithm used in steady-state solvers.

piso: **PISO** algorithm used in transient solvers $\alpha_{\text{CFL}} < 1$.

pimple: **PIMPLE** algorithm used in transient solvers.

DyM: Supports dynamic mesh (eg. mesh refinement, moving meshes).

icoFoam solver

Open the icoFoam.C file:

```
<< find $FOAM_SOLVERS -name "icoFoam.C"
<< less $FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C
```

Solver description:

```
27  Description
28  Transient solver for incompressible, laminar flow of Newtonian fluids.
```

Header files:

- Contain various class definitions.

```
32  #include "fvCFD.H"              // Header file with all the FVM machinery
33  #include "pisoControl.H"        // File related to the PISO algorithm
```

Begining of the main program:

```
37  int main(int argc, char *argv[])
38  {
```

icoFoam solver

Include files:

- Repeated code snippets specific to the case to be solved.

```
39  #include "setRootCase.H"      // Set path and case directories
40  #include "createTime.H"       // Initialise time variable
41  #include "createMesh.H"       // Initialise the mesh to work with
```

Time loop begins:

```
52  while (runTime.loop())
53  {
```

Output results:

```
115  runTime.write();             // Write results to time directories
```

Standard utilities

✓ Over 80+ **OpenFOAM** utilities.

Mesh generation tools:

	Utility	Description
1.	*blockMesh*	Multiblock mesh generator.
2.	*snappyHexMesh*	Automatic split hexahedral mesher which refines and snaps to surface producing hexahedral dominant cells.

Mesh conversion to OpenFOAM format:

	Utility	Description
1.	*ansysToFoam*	**ANSY**S input mesh file exported from I-DEAS.
2.	*gambitToFoam*	**GAMBIT** mesh file.
3.	*gmshToFoam*	Reads .msh **GMSH** file.
4.	*ideasUnvToFoam*	I-DEAS unv format (eg. **SALOME** mesh).

Standard utilities

Mesh manipulation tools:

	Utility	Description
1.	*checkMesh*	Checks the quality of the mesh by providing a detailed output.
2.	*renumberMesh*	Renumbers the cell list to reduce the bandwidth.
3.	*refineMesh*	Refines mesh in multiple directions.
4.	*autoRefineMesh*	Refines cells near to a surface.
5.	*refineHexMesh*	Refines a hexahederal mesh by 2x2x2 cell splitting.

Post-processing tools:

	Utility	Description
1.	*patchAverage*	Calculates average of a field over a patch.
2.	*patchIntegrate*	Calculates integral of a field over a patch.
3.	*probeLocations*	Outputs the field value at a particular location.
4.	*foamCalc*	Utility for simple field calculations at specified times.
5.	*writeCellCentres*	Output cell centres as a volScalarField.

Standard utilities

Parallel processing tools:

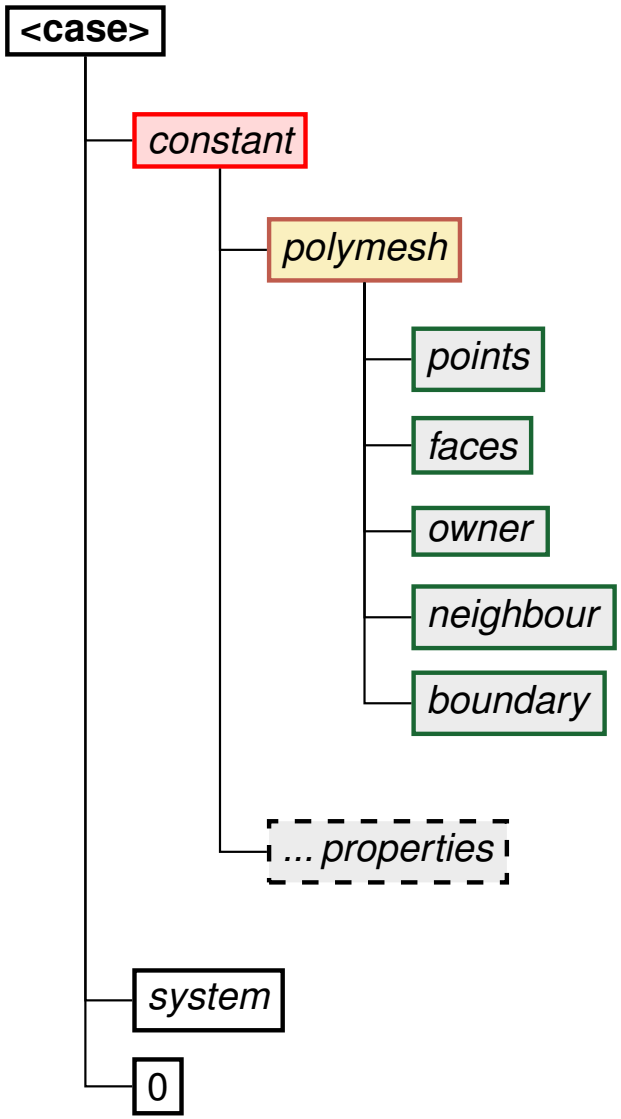
	Utility	Description
1.	*decomposePar*	Decomposes a mesh and fields of a case for parallel execution.
2.	*reconstructParMesh*	Reconstructs the decomposed mesh after parallel execution using only the geometric information.

Scheme of presentation

Contents of this section

We will look at the following:

Directory structure

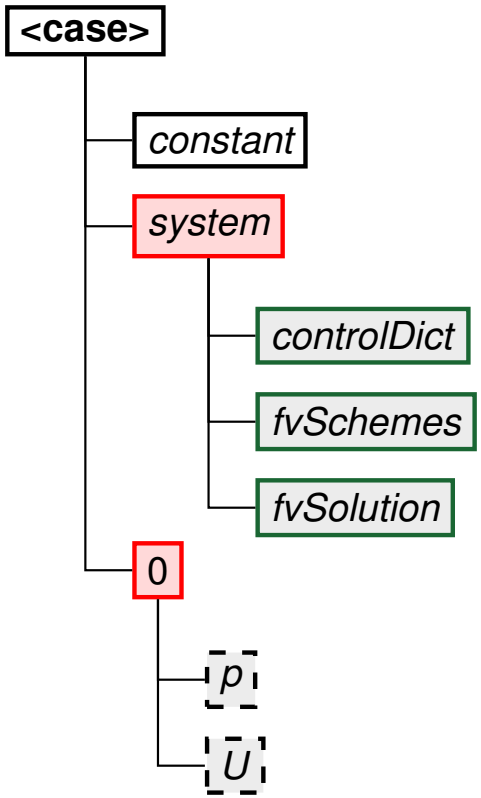


1. *constant*:
   This directory contains the information which remains constant throughout the simulation. It contains the following:

   1.1 *polymesh*:
   Contains all the mesh information including:
   (a) *points* → nodal positions
   (b) *faces* → face connectivities
   (c) *owner* → owner cell labels
   (d) *neighbour* → neighbour cell labels
   (e) *boundary* → boundary information

   1.2 *... properties*:
   Files which specify physical properties for a particular application eg. gravity, viscosity, thermal properties etc.

Directory structure



2. *system*:

   This directory contains all the parameters associated with the solution procedure. It contains at least the following files:

   2.1 *controlDict*:

   Specifies the run control parameters such as start/end time, time step, write interval etc.

   2.2 *fvSchemes*:

   Contains the finite volume discretisation schemes used for the solution procedure such as spatial and temporal discretisations.

   2.3 *fvSolution*:

   Contains equation solvers, algorithm controls and tolerances for the implicit solvers.

3. 0:

   The '0' directory corresponds to zero time. It contains the initial and boundary conditions for variables (ie. pressure p, velocity U) in individual files.

Meshing in OpenFOAM

- **OpenFOAM** only supports **three dimensional meshes** !!!
- 1D & 2D simulations are carried out by using appropriate '*empty*' boundary conditions.
- Supports arbitrary polyhedral cells bounded by arbitrary polygonal faces.

Using blockMesh:

- Simple mesh generator using blocks.
- Allows multiple blocks and curved edges.
- Not suitable to use for complex geometries.

Good practices:

- Run '*checkMesh*' utility to monitor mesh quality.
- Check mesh in **ParaView** before starting the simulation using *paraFoam* command .

Selecting an OpenFOAM solver for simulation

Procedure:

- Select an **OpenFOAM** solver (alias: *sol*) relevant to the type of simulation that you are interested in.
- Copy a suitable tutorial (alias: *tut*) to your run directory (alias: *run*).
- Understand the inputs in that tutorial to run the solver.
- Modify the case according to your needs.

OpenFOAM case file syntax

A *sub-dictionary* is characterised by curly brackets {...}

```
{
.....keyword entries.....
}
```

A *list* is characterised by round brackets (...);

```
List<Type>
<n>
(
.....data entry.....
);
```

A *dimensionSet* is characterised by square brackets [...]

```
[0 1 -1 0 0 0 0] // [Mass Length Time Temp.  Quanity Current Luminosity]
```

OpenFOAM case files: 'system' directory

ControlDict:

```
<< less -N /lordvader/courses/OpenFoam/caseFiles/system/controlDict
```

fvSchemes:

```
<< less -N /lordvader/courses/OpenFoam/caseFiles/system/fvSchemes
```

fvSolution:

```
<< less -N /lordvader/courses/OpenFoam/caseFiles/system/fvSolution
```

OpenFOAM case files: '0' directory

- Initial and boundary conditions must be specified for **all variables** inside '*0*' directory.
- Geometry is broken down into patches where boundary conditions are applied.

Initial and boundary conditions:

```
<< less -N /lordvader/courses/OpenFoam/caseFiles/0/U
```

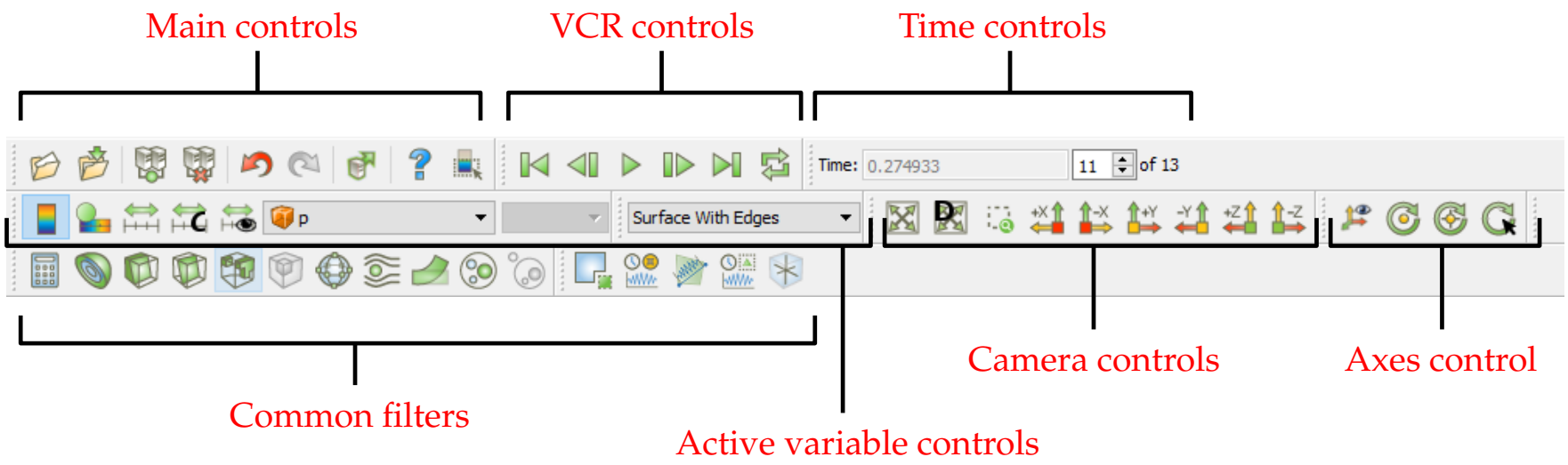
Basic boundary conditions for velocity field:

	BC type	Data	Example
1.	*fixedValue*	value	$\boldsymbol{U} = (5, 10, 0)$
2.	*fixedGradient*	gradient	$\frac{\partial T}{\partial n} = 3.5$
3.	*zeroGradient*	_	$\frac{\partial p}{\partial n} = 0$

How to postprocess OpenFOAM results?

- **OpenFOAM** comes with builtin support for **ParaView**.
- **ParaView** is an open-source, multi-platform data analysis and visualisation application.
- Also possible to visualise results in other third-party softwares.
- **OpenFOAM** results can be visualised using the *paraFoam* utility.

ParaView toolbars:



Running an OpenFOAM tutorial

Default procedure:

- Copy **OpenFOAM** tutorial to you run directory.
  `<< cp -r <tutorial_directory> $FOAM_RUN`
  `<< cd <tutorial_name>`
- Create mesh by using the *blockMesh* command.
  `<< blockMesh`
- Run simulation by using name of an **OpenFOAM** solver (eg. *pisoFoam*).
  `<< pisoFoam`
- Visualise results using the *paraFoam* command.
  `<< paraFoam`

Procedure for complex tutorials:

Some complex tutorials require an *Allrun* script to execute instructions automatically.
`<< find $FOAM_TUTORIALS -name Allrun`

Some tutorials have an *Allclear* script to clean the case directory.

Scheme of presentation

Contents of this section

We will look at the following:

Basic classes

- ✓ Basic classes in **OpenFOAM** have been derived from more fundamental **C++** classes.
- ✓ **OpenFOAM** classes are more advanced and have more functionality.

Comparison of classes:

	C++ class	OpenFOAM class	
1.	int/long	*label*	
2.	bool	*switch*	- Accepts true/false, on/off and yes/no.
3.	string	*word*	
4.	float/double	*scalar*	
5.	—	*vector*	
6.	—	*tensor*	- 3×3 tensor with algebra

GeometricField class

- *GeometricField* class is templated around 3 arguments.
  GeometricField < Type, PatchField, GeoMesh >

A '*Field*' refers to a list of tensors along with a mesh.

- *typedef* is an alias for a class to improve readability.
- Used extensively in **OpenFOAM** particularly in relation to template classes.

The *geometricField< Type >* is renamed used *typedef* declarations to indicate where field variables are stored:

1. *volField<Type>* $\longrightarrow$ Field defined at cell centres.
2. *surfaceField<Type>* $\longrightarrow$ Field defined at cell faces.
3. *pointField<Type>* $\longrightarrow$ Field defined at cell vertices.

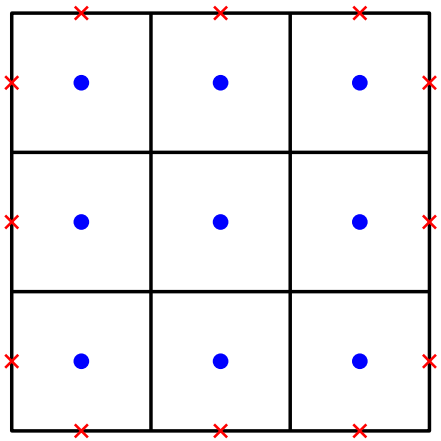
GeometricField class

Internal field:

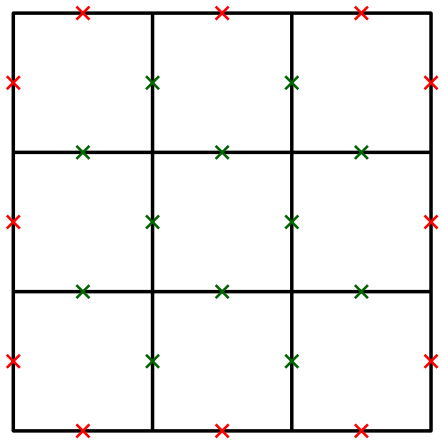
- Defined at cell centers
- Defined at internal faces centers
- Defined at nodes

Boundary field:

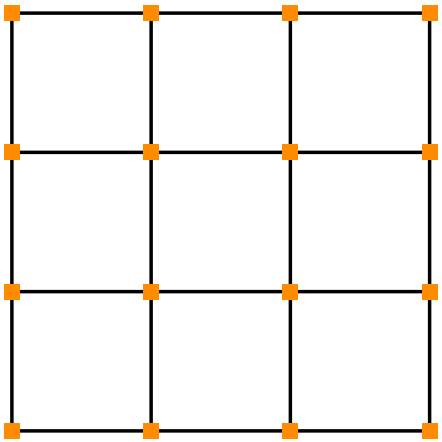
- Defined at boundary face centers



(a) volField



(b) surfaceField



(c) pointField

Equation mimicking

Classical incompressible Navier-Stokes equation:

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) - \mu \nabla^2 \boldsymbol{U} = -\nabla p + \boldsymbol{b}$$

Navier-Stokes equation in OpenFOAM

```
solve
(
    fvm::ddt(rho,U)
    + fvm::div(U,U)
    - fvm::laplacian(mu,U)
    ==
    - fvc::grad(p)
    + b
);
```

1. *typedef - fvm*:
   - Belongs the *finiteVolumeMethod* class.
   - Performs an implicit evaluation and returns an *fvMatrix*.
2. *typedef - fvc*:
   - Belongs to the *finiteVolumeCalculus* class.
   - Performs an explicit calculation and returns a *geometricField*.

Dimensional compatibility

- ✓ **OpenFOAM** is fully dimensional.
- ✓ Fields and properties have dimensions associated with them.
- ✓ **Dimensional checking** is performed to avoid meaningless operations.
- ✓ Units are defined using the *dimensionSet* class.

Dictionary in OpenFOAM

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}

// Thermal diffusivity
DT    DT [0 2 -1 0 0 0 0]   0.01;
```

Sample OpenFOAM code

```
dimensionedTensor sigma
(
    "sigma",
    dimensionSet(1,-1,-2,0,0,0,0),
    tensor(1e6,0,0,0,10e6,0,0,0,5e6)
);
```

Standard dimensional units

	Quantity	SI Base Units						
		kg	m	s	K	mol	A	cd
1.	Density	1	-3	0	0	0	0	0
2.	Linear Momentum	1	-2	-1	0	0	0	0
3.	Pressure	1	-1	-2	0	0	0	0
4.	Force	1	1	-2	0	0	0	0
5.	Lame's Coefficients	1	-1	-2	0	0	0	0
6.	Bulk Modulus	1	-1	-2	0	0	0	0
7.	Young's Modulus of Elasticity	1	-1	-2	0	0	0	0
8.	Specific Heat Capacity	0	2	-2	-1	0	0	0
9.	Thermal Conductivity	1	1	-3	-1	0	0	0

Tensor operations

- Tensor operations available to all **OpenFOAM** *tensor* classes

Product operations:

	Product operator	**OpenFOAM** notation	Rank of a & b
1.	Outer	$a * b$	≥ 1
2.	Inner/dot	$a \,\&\, b$	≥ 1
3.	Double inner/dot	$a \,\&\&\, b$	≥ 2
4.	Cross	$a \hat{} b$	$= 1$

Other operations:

	Product operator	**OpenFOAM** notation
1.	Magnitude	$mag\,(a)$
2.	Square of magnitude	$magSqr\,(a)$
3.	Square	$sqr\,(a)$
4.	Power	$pow\,(a, n)$

Tensor operations

Operations exclusive to tensors of rank 2:

	Operator	**OpenFOAM** notation
1.	Transpose	$A \cdot T()$
2.	Diagonal	$diag\,(A)$
3.	Trace	$tr\,(A)$
4.	Symmetric part	$symm\,(A)$
5.	Skew-symmetric part	$skew\,(A)$
6.	Determinant	$det\,(A)$
7.	Cofactor	$cof\,(A)$
8.	Inverse	$inv\,(A)$

Operations exclusive to tensors of rank 0:

	Operator	**OpenFOAM** notation
1.	Square root	$sqrt\,(s)$
2.	Exponential	$exp\,(s)$
3.	Natural log	$log\,(s)$
4.	Log base 10	$log10\,(s)$
5.	Sine	$sin\,(s)$
6.	Cosine	$cos\,(s)$
7.	Tangent	$tan\,(s)$

Mesh parameters

Important mesh parameters:

	Description	Access function	Data type
1.	Total number of cells	*mesh.nCells()*	*scalar*
2.	Total number of internal faces	*mesh.nInternalFaces()*	*scalar*
3.	Cell volumes	*mesh.V()*	*volScalarField*
4.	Face area normal vectors	*mesh.Sf()*	*surfaceVectorField*
5.	Face area magnitudes	*mesh.magSf()*	*surfaceVectorField*

Coordinates:

	Coordinates	Access function	Data type
1.	Cell center	*mesh.C()*	*volVectorField*
2.	Face center	*mesh.Cf()*	*surfaceVectorField*
3.	Nodal	*mesh.points()*	*pointField*

Mesh connectivities

- All mesh connectivities are stored in *labelListList* data type.

	Connectivity			Access function
1.	Node	$\longrightarrow$	Node	*mesh.pointPoints()*
	Node	$\longrightarrow$	Edge	*mesh.edgePoints()*
	Node	$\longrightarrow$	Face	*mesh.pointFaces()*
	Node	$\longrightarrow$	Cell	*mesh.pointCells()*
2.	Edge	$\longrightarrow$	Node	*mesh.edges()*
	Edge	$\longrightarrow$	Face	*mesh.edgeFaces()*
3.	Face	$\longrightarrow$	Node	*mesh.faces()*
	Face	$\longrightarrow$	Edge	*mesh.faceEdges()*
4.	Cell	$\longrightarrow$	Face	*mesh.cells()*
	Cell	$\longrightarrow$	Cell	*mesh.cellCells()*

wmake compilation tool

- ✓ **OpenFOAM** uses its own compilation tool *wmake*.
- ✓ **OpenFOAM** applications and libraries require a '*Make*' directory.

Compilation is performed by executing the '*wmake*' command from the directory containing the Make folder.

Scheme of presentation

Contents of this section

We will perform the following tutorials:

Create a mesh using blockMesh utility

Inputs for creating a mesh using *blockMesh* requires a *constant/polymesh/blockMeshDict* file.

Copy blockMesh tutorial:

```
<< cp -r /lordvader/courses/OpenFoam/tutorials/blockMesh/ $FOAM_RUN
<< cd $FOAM_RUN/blockMesh
<< tree
```

Run blockMesh command:

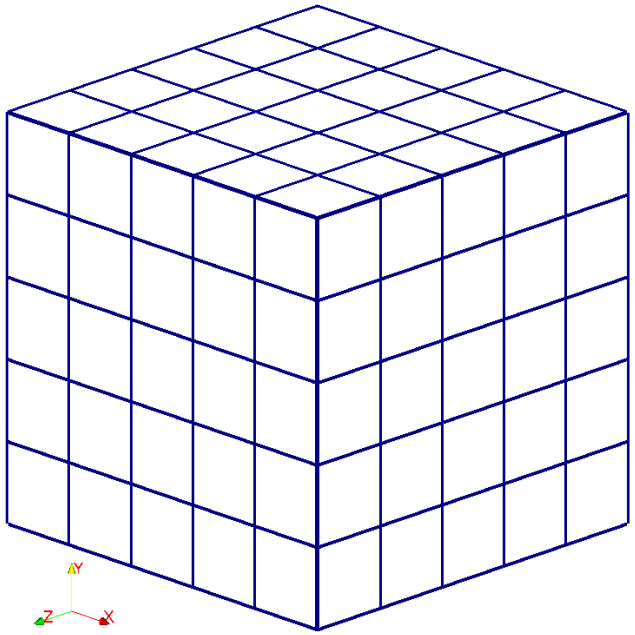
```
<< pwd
<< blockMesh
<< tree
```

Check the mesh quality:

```
<< checkMesh
```

Visualise the mesh:

```
<< paraFoam
```



Study blockMeshDict file

Open blockMeshDict file:

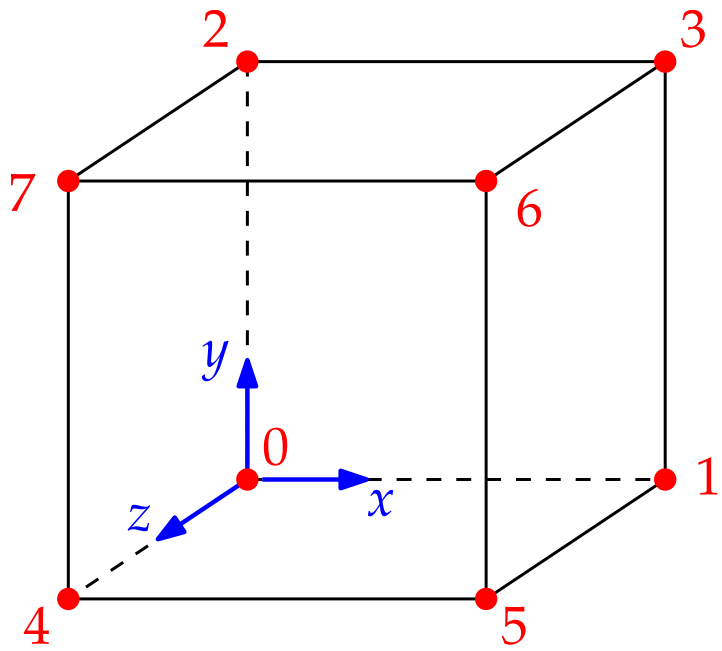
```
<< less -N constant/polyMesh/blockMeshDict
```

Convert units:

```
17 convertToMeters 0.001;      // Scaling to convert to millimeters
```

List of vertices:

```
19 vertices
20 (
21     (0 0 0)      // Vertex 0
22     (1 0 0)      // Vertex 1
23     (0 1 0)      // Vertex 2
24     (1 1 0)      // Vertex 3
25     (0 0 1)      // Vertex 4
26     (1 0 1)      // Vertex 5
27     (1 1 1)      // Vertex 6
28     (0 1 1)      // Vertex 7
29 );
```

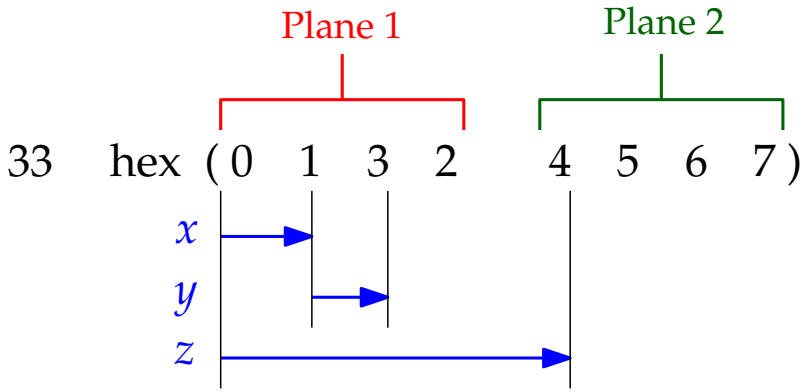


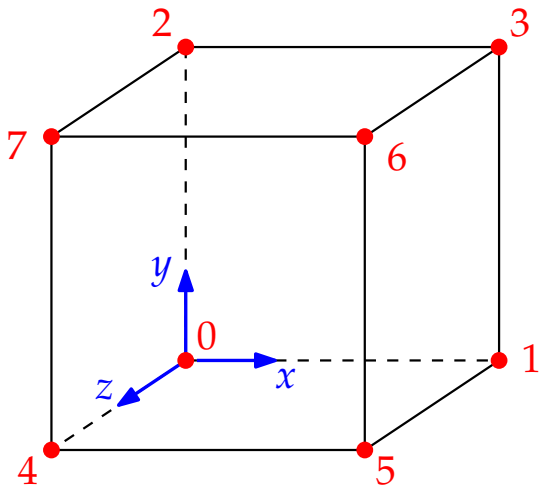
Study blockMeshDict file

Defining cells:

```
31 blocks
32 (
33     hex (0 1 3 2 4 5 6 7)      // Vertex list for creation of a hexahedral cell
34     (5 5 5)                    // No. of cells in each direction
35     simpleGrading (1 1 1)      // Cell expansion ratios
36 );
```

Ordering of vertices is critical when defining blocks!





Study blockMeshDict file

Boundary field:

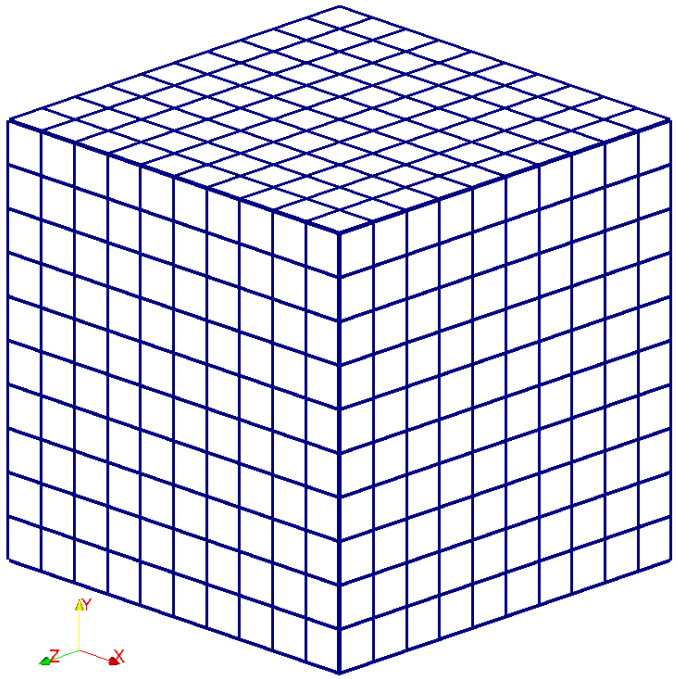
```
38 boundary                    // Contains definition of boundary patches
39 (
        .....
67 );
```

Boundary patch:

```
40 sides                       // Boundary patch name
41 {
42     type patch;
43     face
44     (
45         (4 0 2 7)           // Boundary face definitions
46         (1 5 6 3)
47         (0 1 3 2)
48         (4 5 6 7)
49     );
50 }
```

Exercise

Modify the *blockMeshDict* to increase mesh density to $10 \times 10 \times 10$ cells.



You can edit the file using any text editor (eg. gedit, vim, nano).

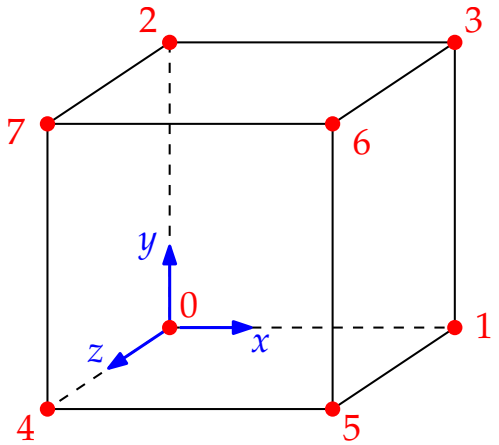
<< `gedit constant/polyMesh/blockMeshDict`

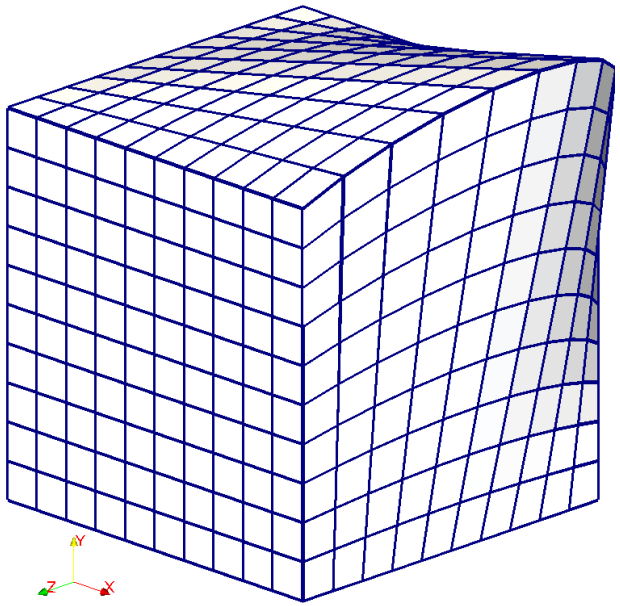
Exercise

Connect points with an **arc** instead of a line.

Add the following after the blocks definition:

```
38 edges
39 (
40     arc 6 3 (1.3 1.3 0.5)
41 );
```





What have we learnt in this tutorial!

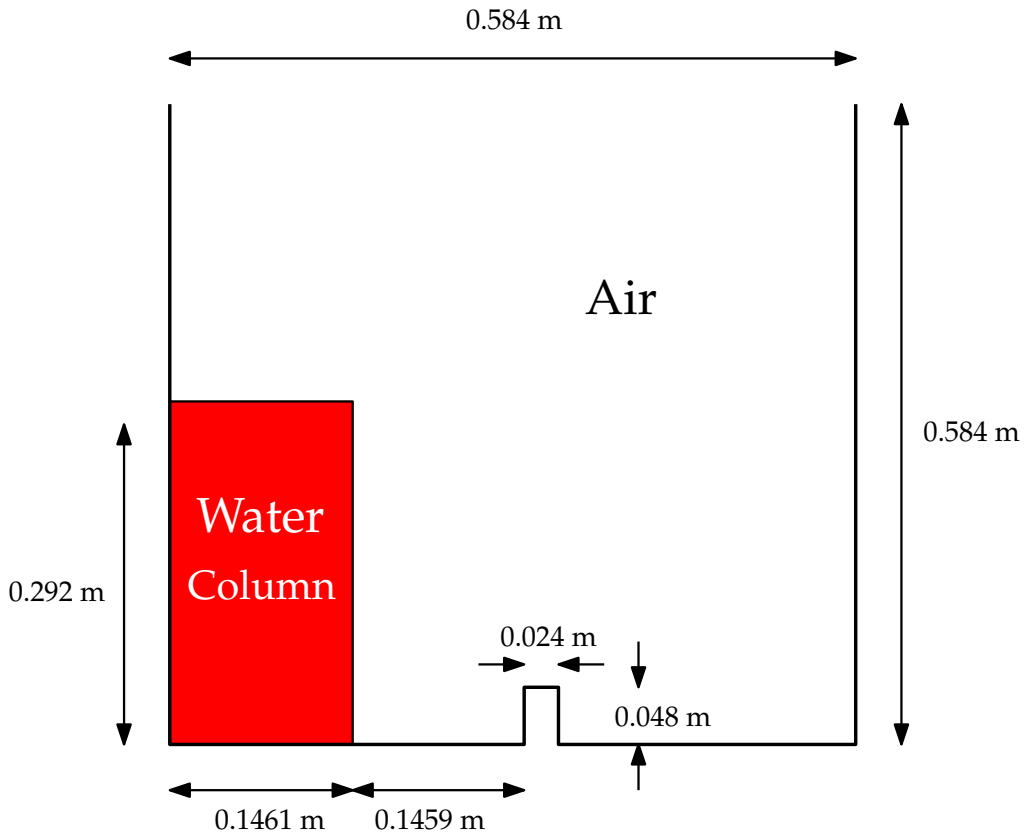
- ✓ Understand the structure of blockMeshDict file.
- ✓ Modify blockMeshDict file.
- ✓ Use *blockMesh* utility to generate meshes.
- ✓ Use *checkMesh* utility to determine the quality of mesh.
- ✓ Use *paraFoam* utility to visualise the mesh before a simulation.

Two phase flow tutorial

interFoam solver based on the Volume Of Fluid (**VOF**) interface capturing approach.

Problem description:

- Transient two phase flow
- Incompressible, isothermal flow
- Immiscible fluids



interFoam solver

Copy damBreak tutorial:

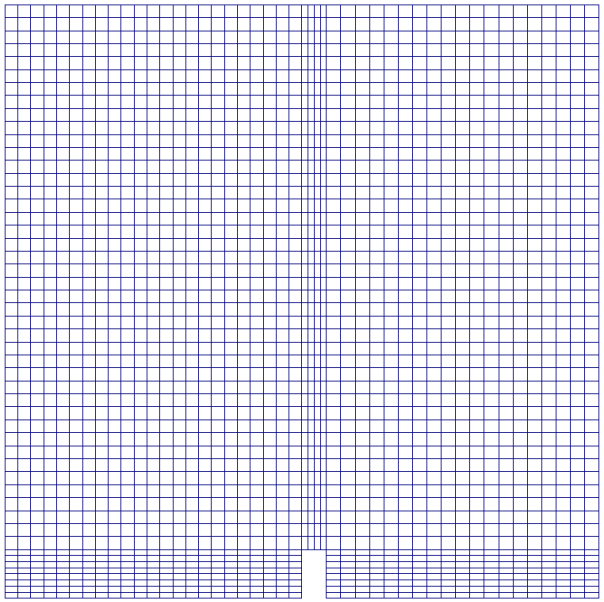
```
<< cp -r $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak/ $FOAM_RUN
```

Change to damBreak directory:

```
<< cd $FOAM_RUN/damBreak
<< tree
```

Create and view mesh:

```
<< blockMesh
<< checkMesh
<< paraFoam
```



Run the solver:

```
<< interFoam
```

```
-> FOAM FATAL IO ERROR:
cannot find file
file:  /home/jibran/OpenFOAM/jibran-2.4.0/run/damBreak/0/alpha.water at line 0.
From function regIOobject::readStream()
in file db/regIOobject/regIOobjectRead.C at line 73.
FOAM exiting
```

interFoam solver

Create alpha.water file:

```
<< cp -r 0/alpha.water.org 0/alpha.water
<< tree
```

Run the solver again:

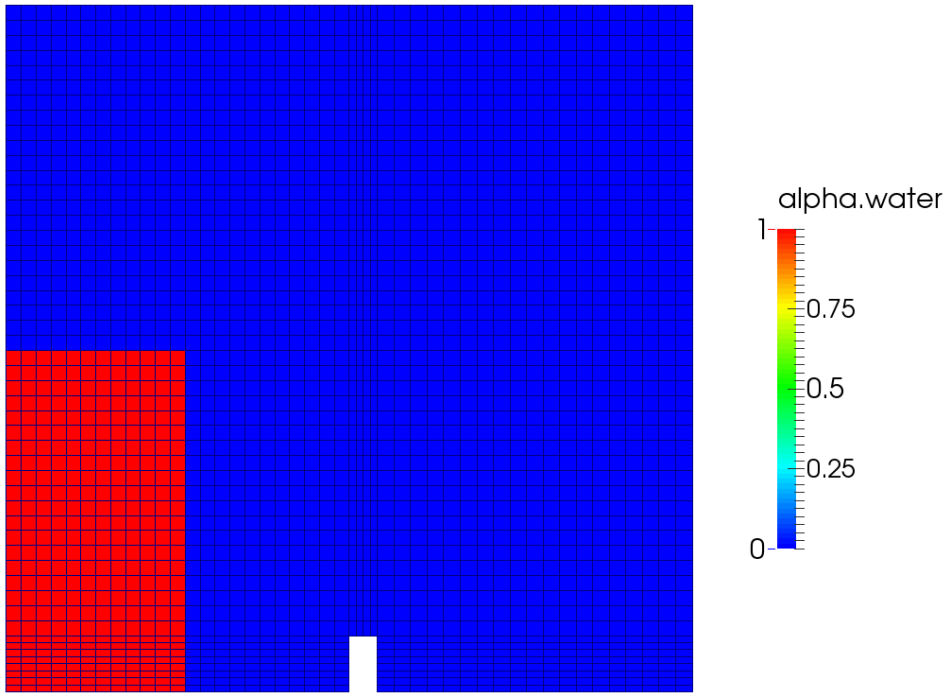
```
<< interFoam > log
<< ls
<< less -N log
```

Clear time directories:

```
<< pyFoamClearCase.py .
<< ls
<< less PyFoamHistory
```

Initialise the alpha.water field:

```
<< less system/setFieldsDict
<< setFields
<< paraFoam
```



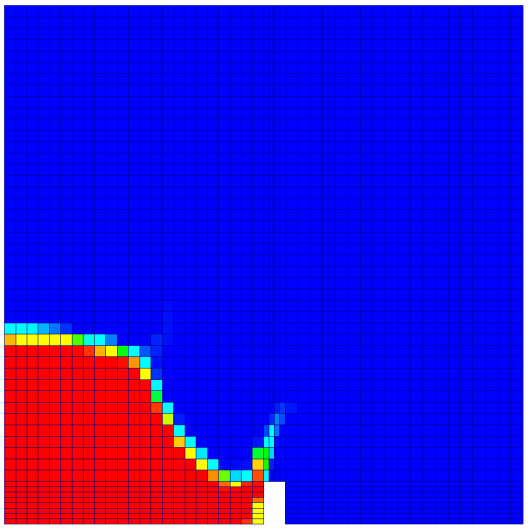
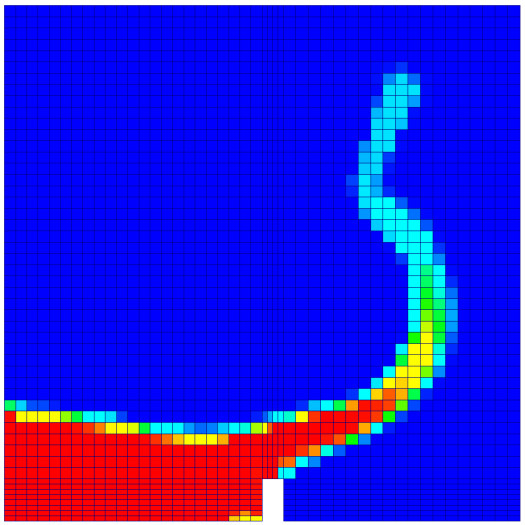
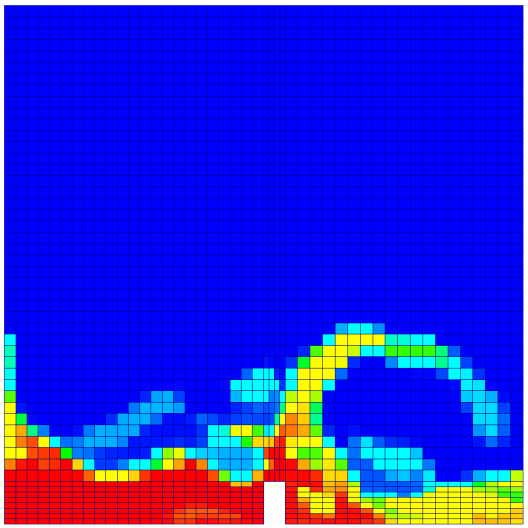
interFoam solver

Check 0/alpha.water:

```
<< head -35 0/alpha.water
```

Run the solver successfully:

```
<< interFoam | tee log
<< less log
<< paraFoam
```

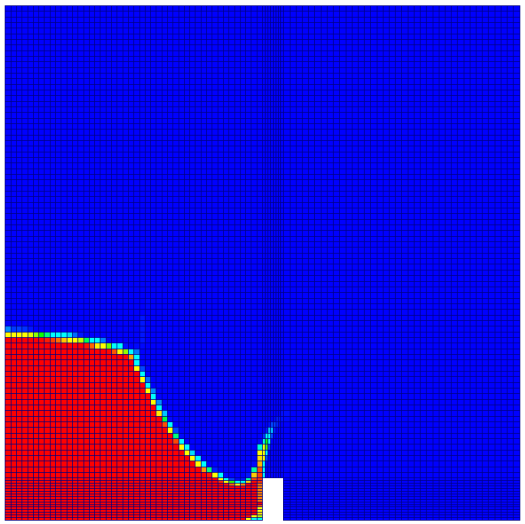
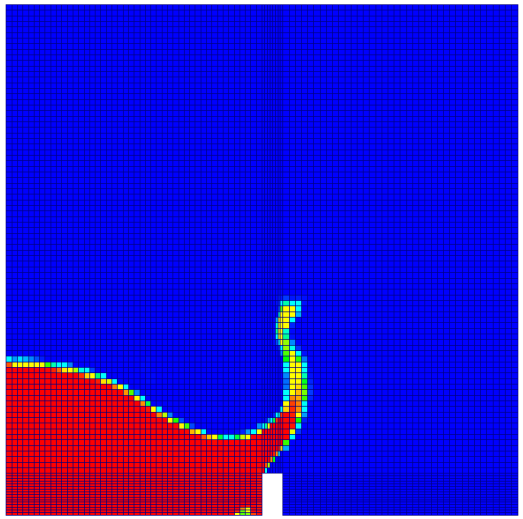
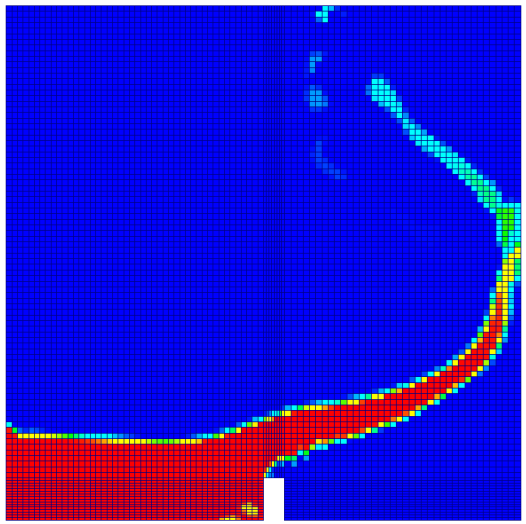




Exercise

Run the same problem with a refined mesh and save an animation.

Hints:

- A utility "*refineMesh*" could be utilised
- Please pay attention to the error messages displayed on your terminal.
  The solution is always there!!!

What have we learnt in this tutorial!

- ✓ Understand **OpenFOAM** errors displayed on the terminal.
- ✓ Use *setFields* utility to initialise fields.
- ✓ Running an **OpenFOAM** solver.
- ✓ Output **OpenFOAM** solver data to a log file.
- ✓ Use *pyFoamClearCase.py .* to clear time directories.
- ✓ Use *paraFoam* utility to visualise results.
- ✓ Use *refineMesh* utility for mesh refinement.

In this tutorial we will learn the usage of some important **OpenFOAM** utilities.

fluentMeshToFoam utility

How to use a mesh generated from **ANSYS FLUENT** which is to be solved using an **OpenFOAM** solver?

Search for a suitable tutorial:

```
<< grep -r -n fluentMeshToFoam $FOAM_TUTORIALS
```

Copy the elbow tutorial:

```
<< run
<< cp -r $FOAM_TUTORIALS/incompressible/icoFoam/elbow/ .
```

Change to elbow directory:

```
<< cd elbow
<< tree
```

Use the fluentMeshToFoam utility:

```
<< fluentMeshToFoam elbow.msh
<< fluentMeshToFoam -help
<< checkMesh
```

foamToVTK utility

Convert to ParaView's VTK format:

<< **foamToVTK**

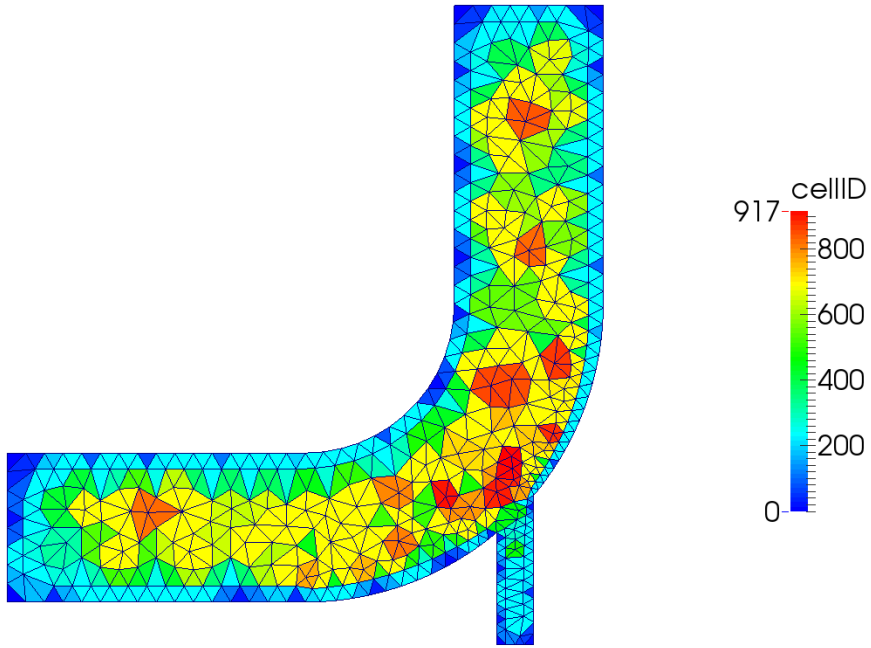
<< **tree**

Read mesh in paraview:

<< **paraview**

File ⟶ Open ⟶ "VTK/elbow_0.vtk"

Display surface with edges and color by cellID.



✗ High bandwidth slows the performance of matrix solvers.

✓ Renumbering of cell labels can reduce the bandwidth in a solution matrix.

renumberMesh utility

Renumber cell labels :

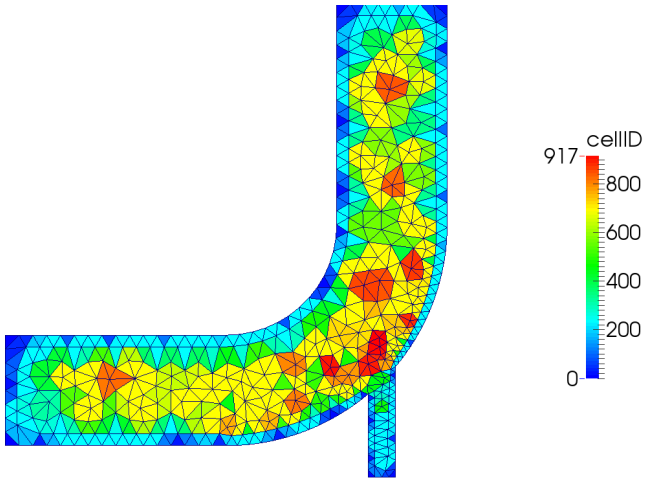
```
<< renumberMesh
<< renumberMesh -overwrite
```

Display new cell labels:

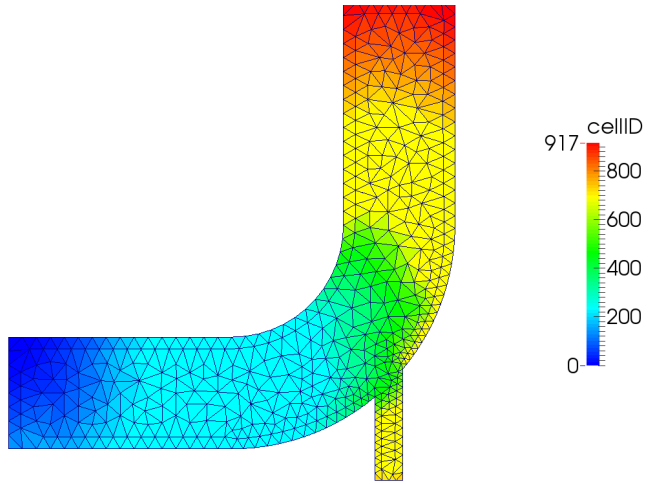
```
<< foamToVTK
<< paraview&
```

File ⟶ Open ⟶ "VTK/elbow_0.vtk"

Display surface with edges and color by cellID.



(a) Before



(b) After

What have we learnt in this tutorial!

- ✓ Search for a particular text using 'grep' command in **OpenFOAM** installation directory.
- ✓ See usage of an **OpenFOAM** command using '-help' option.
- ✓ Convert mesh from **FLUENT** to **OpenFOAM** format using the *fluentMeshToFoam* utility.
- ✓ Convert mesh from **OpenFOAM** to **VTK** format using the *foamToVTK* utility.
- ✓ Use *renumberMesh* utility to decrease bandwidth.

Perform a serial run using a fine mesh of the damBreak tutorial and compare it against results from a **parallel** run.

Serial run

Copy damBreakFine case files:

```
<< run
<< cp -r /lordvader/courses/OpenFoam/tutorials/damBreakFine .
<< mv damBreakFine damBreakFine_serial
<< cp -r damBreakFine_serial damBreakFine_parallel
<< cd damBreakFine_serial
<< tree
```

Run the problem:

```
<< blockMesh
<< checkMesh
<< setFields
<< interFoam | tee log_serial
```

Visualise resuls:

```
<< paraFoam
```

Simulation time for a serial run: ≈ 223 sec.

Step 1: Domain decomposition

- **OpenFOAM** uses domain decomposition for parallel runs.
- Geometry is broken down and allocated to separate processors.
- Decomposition requires a *system/decomposeParDict* file.
- Decomposition process is initiated by using the *decomposePar* utility.

Step 1: Domain decomposition

Check contents of decomposeParDict:

```
<< cd $FOAM_RUN/damBreakFine_parallel
<< less -N system/decomposeParDict
```

```
18 numberOfSubdomains    9;              // No. of subdomains for decomposition
19
20 method                 simple;         // Method of decomposition
21
22 simpleCoeffs                           // Coefficients for simple method
23 {
24     n                  ( 3 3 1 );      // Domain split in x,y,z directions
25     delta              0.001;          // Cell skew factor (Default value = 0.001)
26 }
27
28 hierarchicalCoeffs                     // Coefficients for hierarchical method
29 {
30     n                  ( 1 1 1 );
31     delta              0.001;
32     order              xyz;            // Order in which the decomposition is done
33 }
34
35 distributed          no;               // If data is distributed over several disks
```

Step 1: Domain decomposition

Find available processors:

```
<< grep processor /proc/cpuinfo
```

Modify decomposeParDict:

```
<< gedit system/decomposeParDict
```

Modify the 'numberOfSubdomains' and the domain split 'n' accordingly:

```
18 numberOfSubdomains    4;              // No. of subdomains for decomposition
19
20 method                 simple;         // Method of decomposition
21
22 simpleCoeffs                           // Coefficients for simple method
23 {
24     n                 ( 2 2 1 );      // Domain split in x,y,z directions
25     delta             0.001;          // Cell skew factor (Default value = 0.001)
26 }
```

Some decomposition methodologies:

1. simple $\longrightarrow$ Used for simple geometries by using geometric decomposition.
2. hierarchical $\longrightarrow$ Similar to simple but with order of direction split.
3. scotch $\longrightarrow$ Used for complex geometries by minimising the number of processor boundaries.
4. manual $\longrightarrow$ User specified decomposition for each cell.

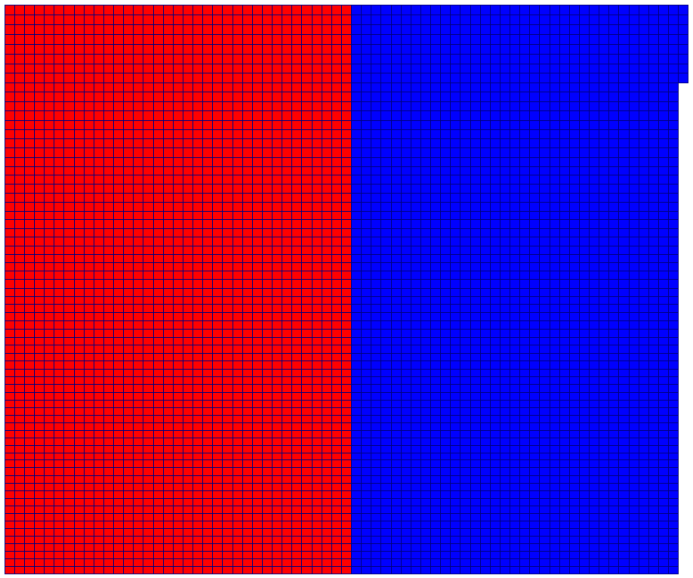
Step 1: Domain decomposition

Initialise the problem:

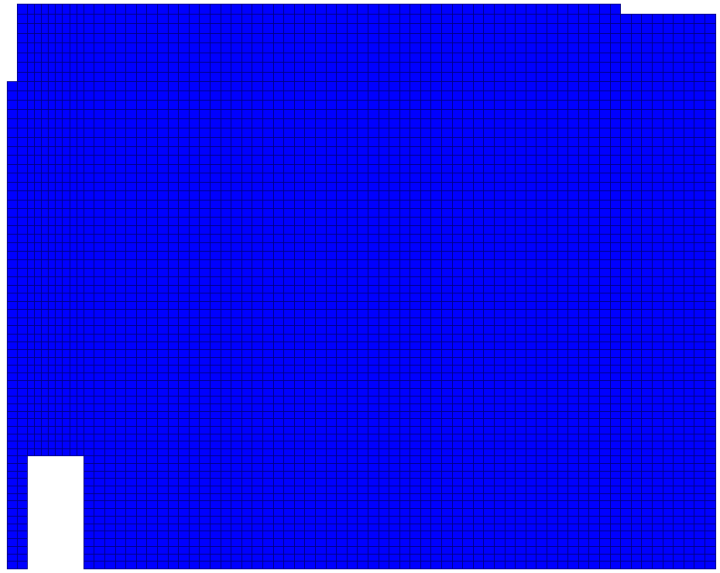
```
<< blockMesh
<< setFields
<< checkMesh
```

Decompose domain for parallel execution:

```
<< decomposePar
<< tree
<< paraFoam -case processor0
```



(a) Processor 0



(b) Processor 1

Step 2: Parallel run

- **OpenFOAM** uses the public domain **openMPI** implementation of the standard Message Passing Interface (**MPI**).
- Syntax for parallel execution on a single node is as follows:
  `<< mpirun -np <no_of_processors> <solver_name> -parallel`

Parallel run:

```
<< mpirun -np 4 interFoam -parallel > log_parallel&
<< gnome-system-monitor
```

Visualise results:

```
<< ls
<< tree processor0
<< paraFoam -case processor0
```

Simulation time for parallel run:

```
<< tail log_parallel
```

	No. of processors	Simulation time (s)
1.	1	≈ 223
2.	4	≈ 86
3.	6	≈ 91
4.	8	≈ 96

Step 3: Reconstruction

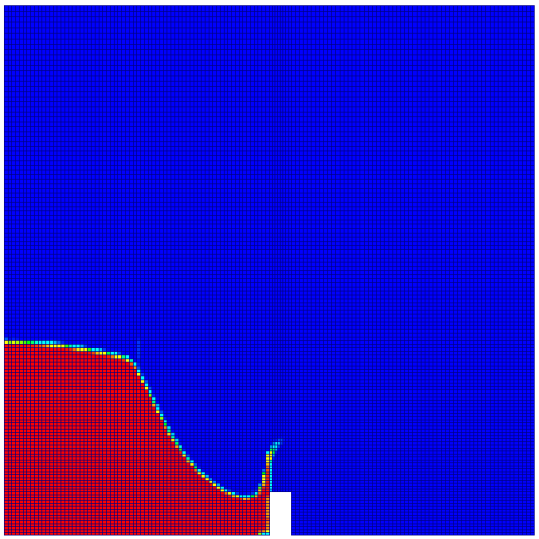
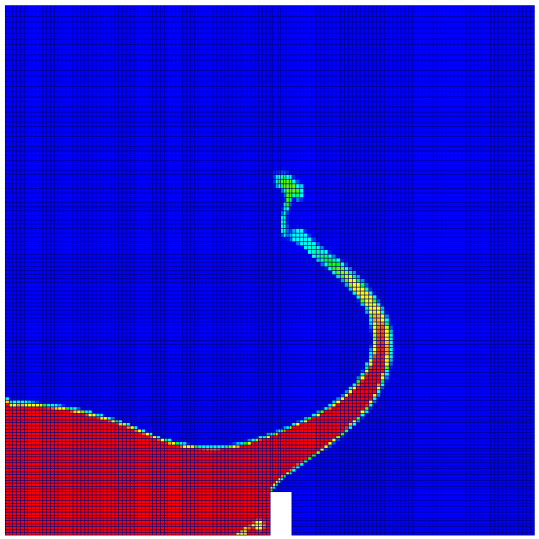
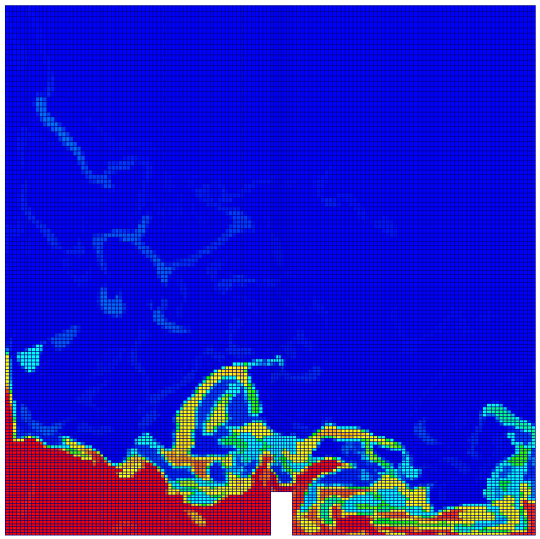
reconstructPar utility can be used to reassemble the fields and mesh from the decomposed parallel run.

Reconstruction of results:

```
<< reconstructPar
```

Visualise results:

```
<< ls
<< paraFoam
```

What have we learnt in this tutorial!

- ✓ Setup *decomposeParDict* with appropriate number of processors.
- ✓ Use *decomposePar* utility to decompose to domain.
- ✓ Launch a parallel run on a single node using mpirun with '-parallel' option.
- ✓ Use *reconstructPar* utility to reassemble decomposed fields and mesh.

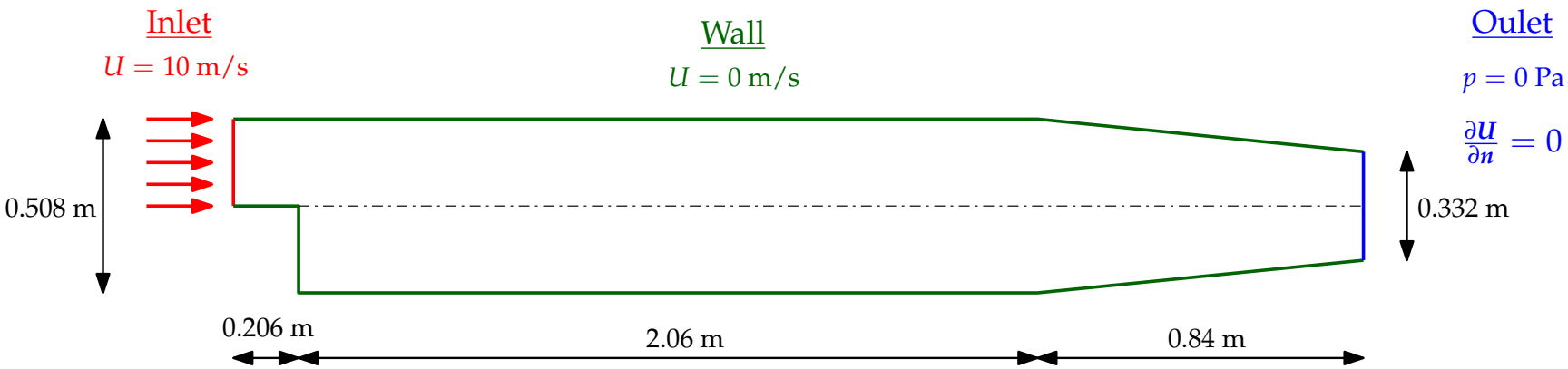
Exercise

Simulate the problem using an appropriate OpenFOAM solver.

Problem description:

- Two dimensional, steady state flow.
- Incompressible and turbulent flow.
- Newtonion fluid with viscosity $\nu = 1 \times 10^{-5} m^2/s$.

Case setup:



Exercise

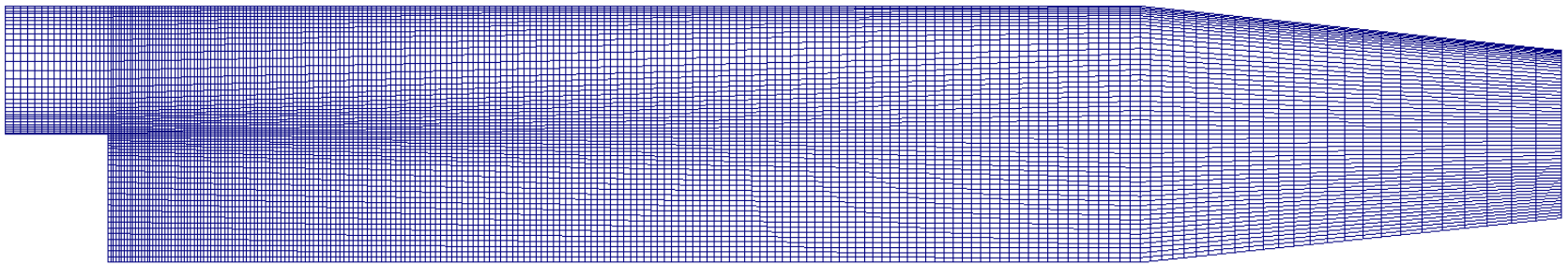
Copy the simulation case:

```
<< cp -r /lordvader/courses/OpenFoam/tutorials/pitzDaily/ $FOAM_RUN
```

Hints:

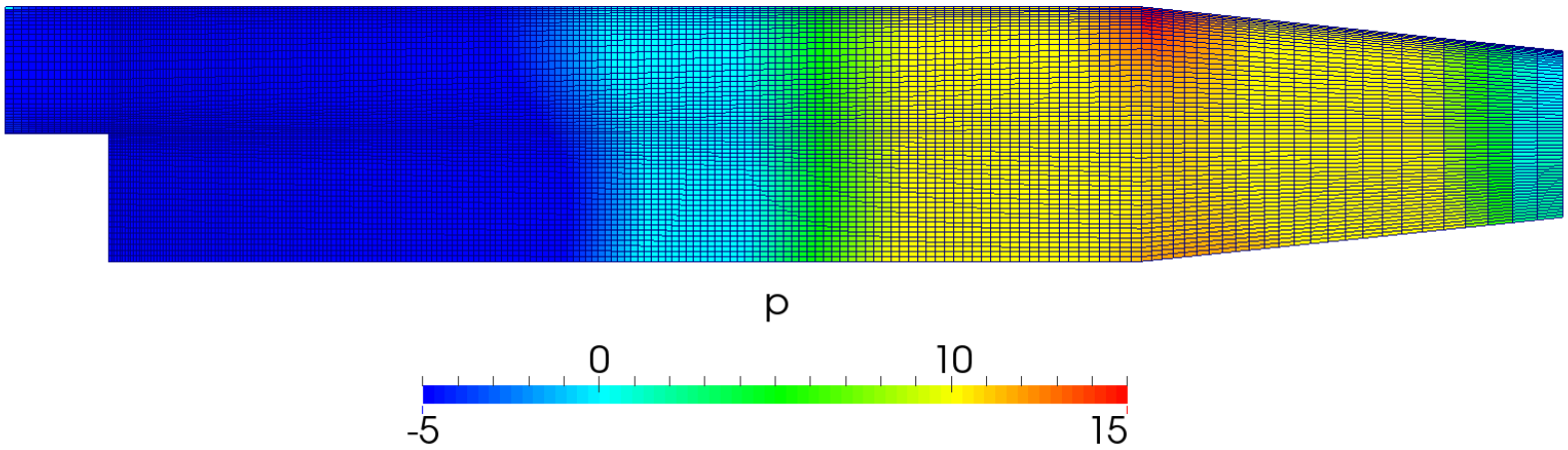
- Please pay attention to the error messages displayed on your terminal.
- Don't start writing files from scratch for missing files. **Be smart!**.
- Remember to define the problem variables correctly.
- The geometry is provided in the simulation case.
- Parameters associated to turbulence modelling are already defined.

Geometry and meshing:

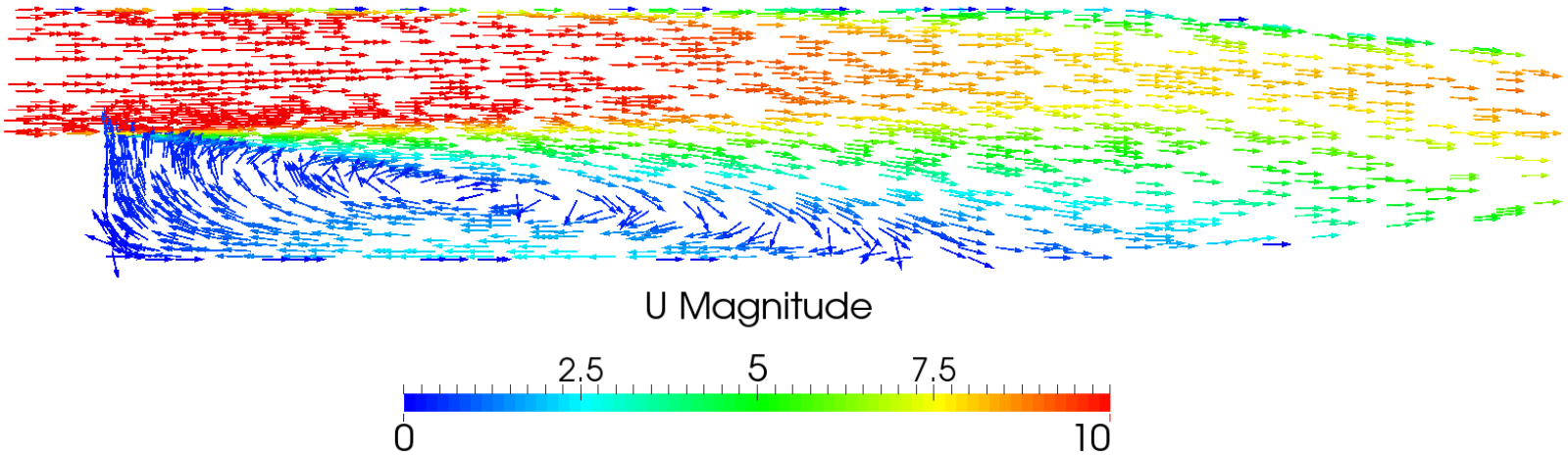


Results at time (817 sec):

Pressure distribution:



Velocity vectors:



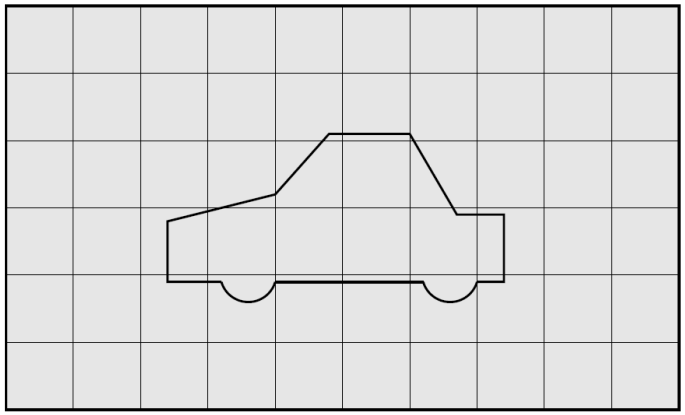
What have we learnt in this tutorial!

- ✓ Select an **OpenFOAM** solver based on our problem definition.
- ✓ Understand additional **OpenFOAM** errors displayed on the terminal.
- ✓ Modify parameters in the input files according to our needs.
- ✓ Create input files by copying an already existing similar file.
- ✓ Use the glyph filter in **ParaView** to plot vectors.

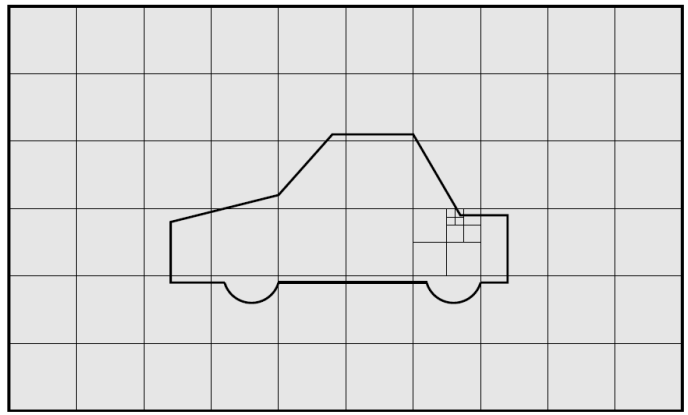
Basics of snappyHexMesh

Steps for creating a snappyHexMesh:

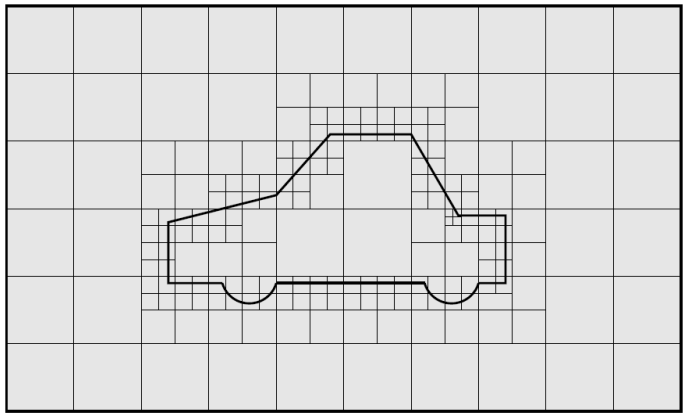
1. Creation of a castellated mesh



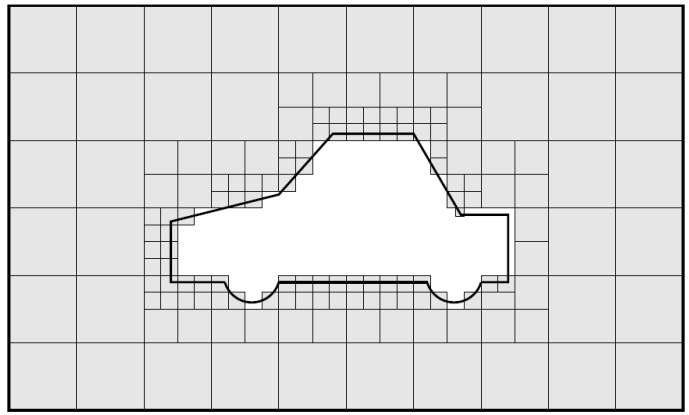
(a) Background mesh



(b) Refinement initiated



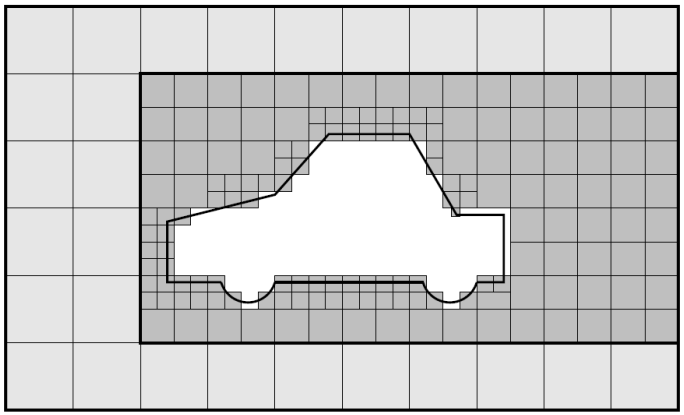
(c) Refinement completed



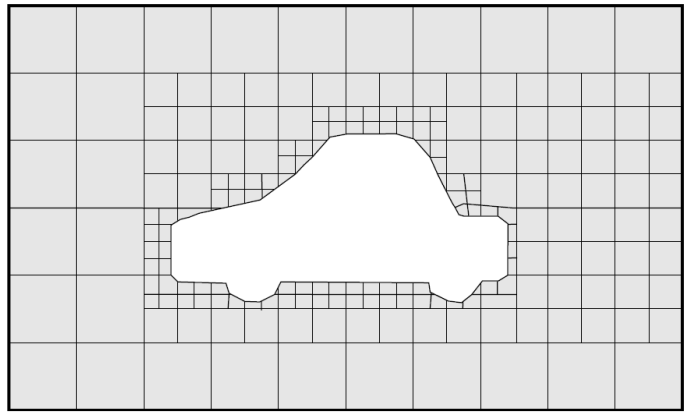
(d) Cell removal

Basics of snappyHexMesh

2. Snap to surface

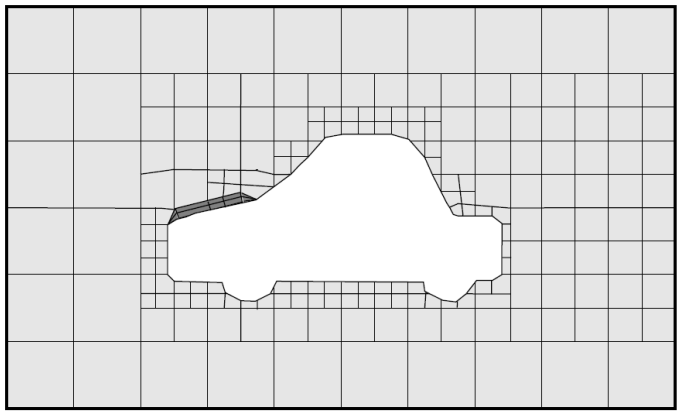


(e) Region refinement



(f) Snapping

3. Addition of layers



(g) Layer addition

Surface meshes

Copy the motorbike tutorial:

```
<< run
<< cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/motorBike .
<< cd motorike
<< tree
<< less constant/trisurface/README
```

Obtain motorbike geometry:

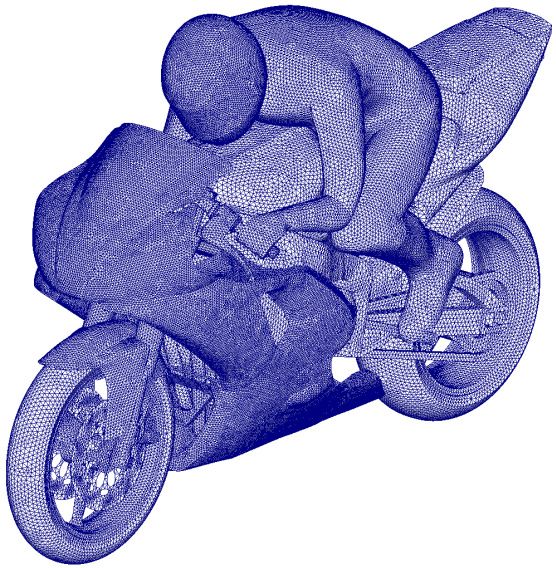
```
<< cd constant/triSurface
<< cp -r $FOAM_TUTORIALS/resources/geometry/motorBike.obj.gz .
<< gunzip motorBike.obj.gz
```

Visualise geometry:

```
<< paraview
```

File $\longrightarrow$ Open $\longrightarrow$ "constant/triSurface/motorBike.obj"

Display surface with edges



Surface meshes

Utilities for surface meshes:

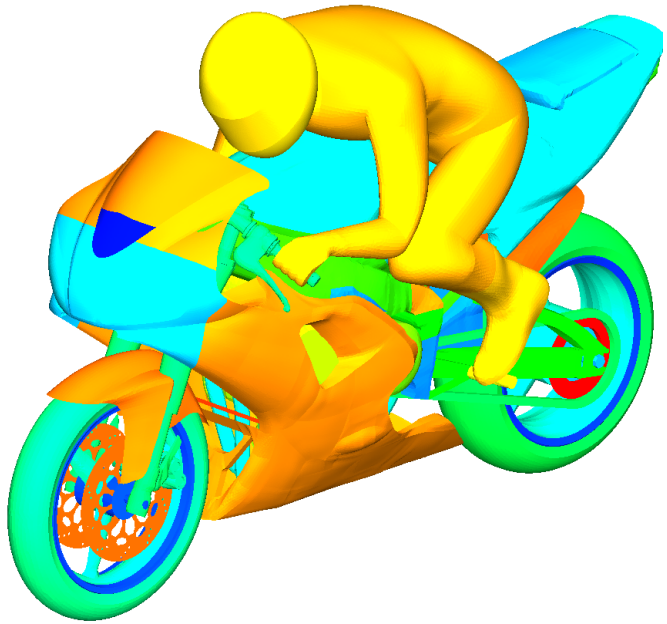
```
<< cd ../..
<< surface[TAB][TAB]
<< surfaceMeshInfo -help
<< surfaceMeshInfo constant/triSurface/motorBike.obj
    nPoints :  132871
    nFaces  :  331653
    area    :  12.1486
```

Extract surface features from the geometry:

```
<< gedit system/surfaceFeatureExtractDict
    28 includedAngle 180
<< surfaceFeatureExtract
<< tree
```

View patches:

```
<< surfaceConvert
    constant/triSurface/motorBike.obj
    constant/triSurface/motorBike.vtk
```



Castellated mesh

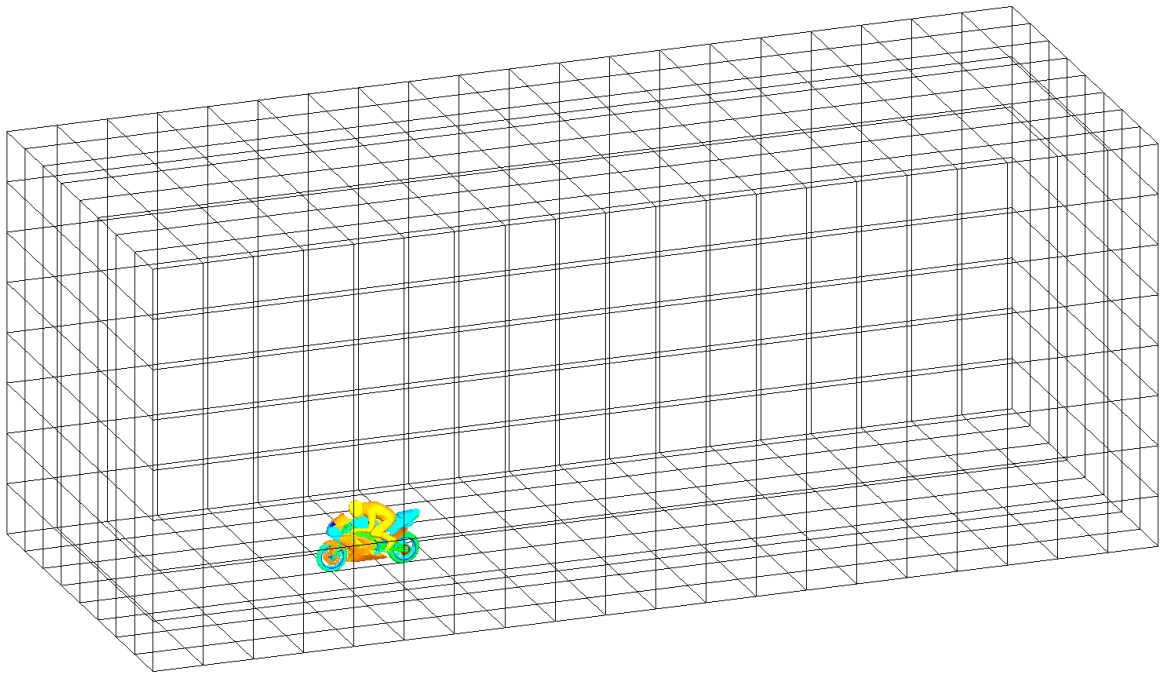
Meshing using *snappyHexMesh* utility requires a base background mesh to start with.

Notes:

- Must only consist of hexahedral cells.
- The aspect ratio of cells must be equal to **1** for optimum behaviour.

Create background mesh:

<< **blockMesh**

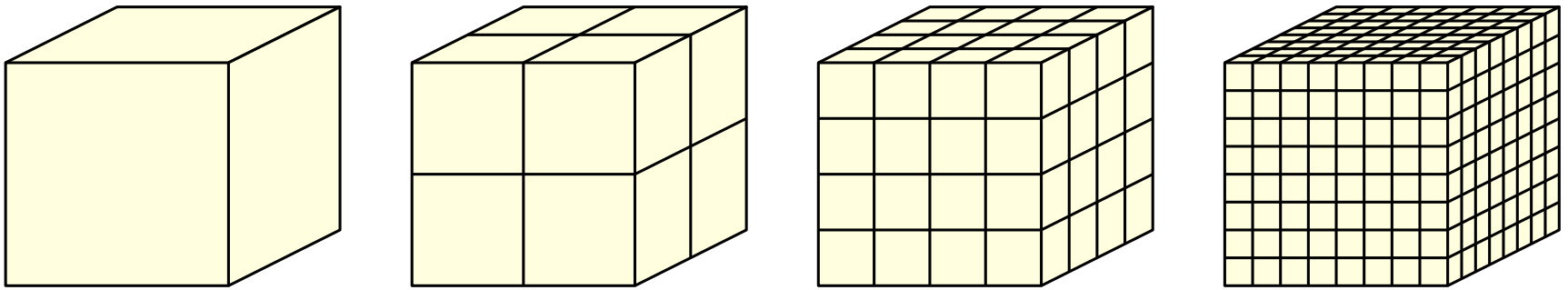


Castellated mesh

Run snappyHexMesh in parallel:

```
<< decomposePar
<< foamJob -parallel -screen snappyHexMesh
<< less system/snappyHexMeshDict
```

Local mesh refinement:

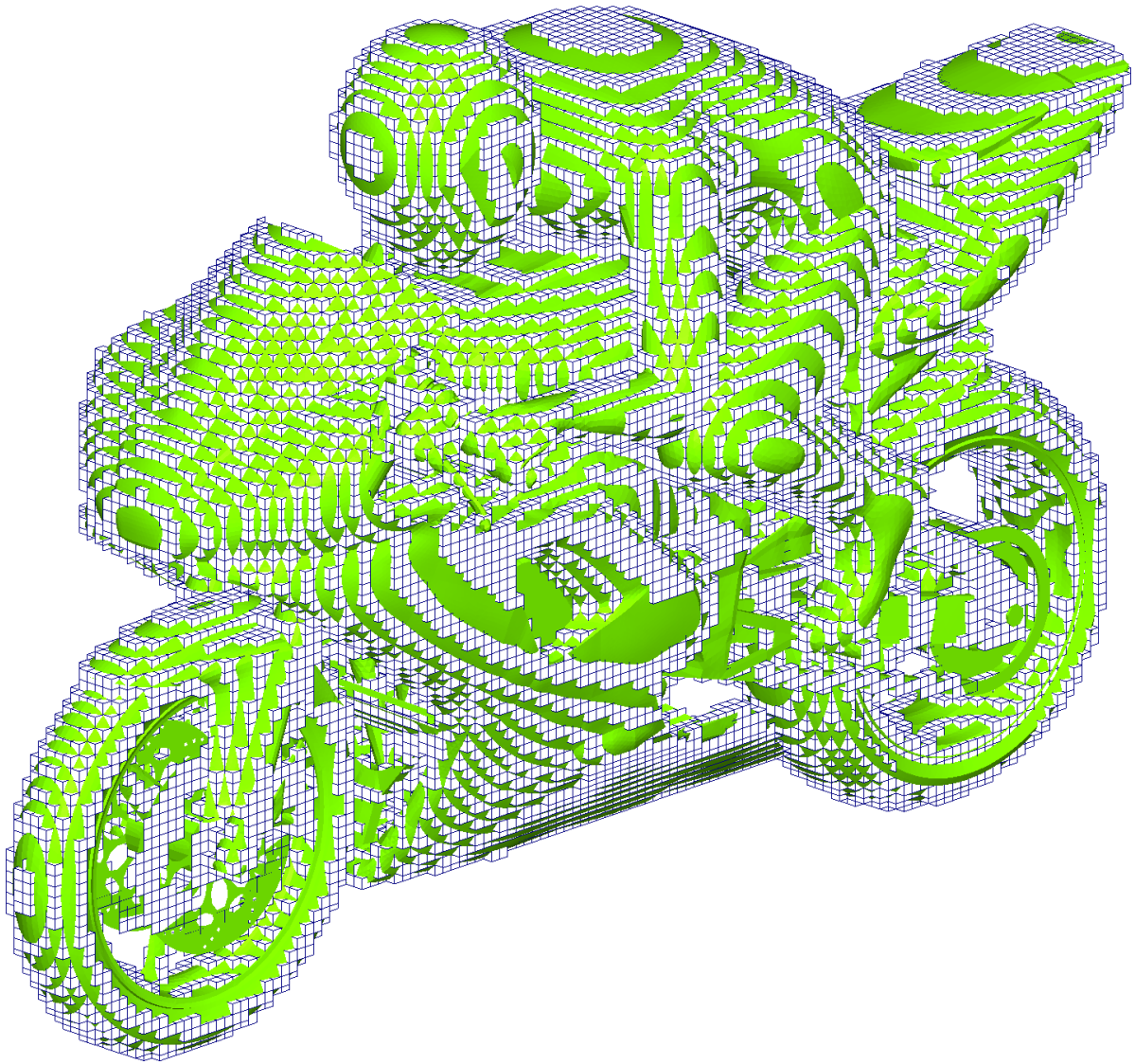


(a) Level 0 (b) Level 1 (c) Level 2 (d) Level 4

Reconstruct decomposed mesh:

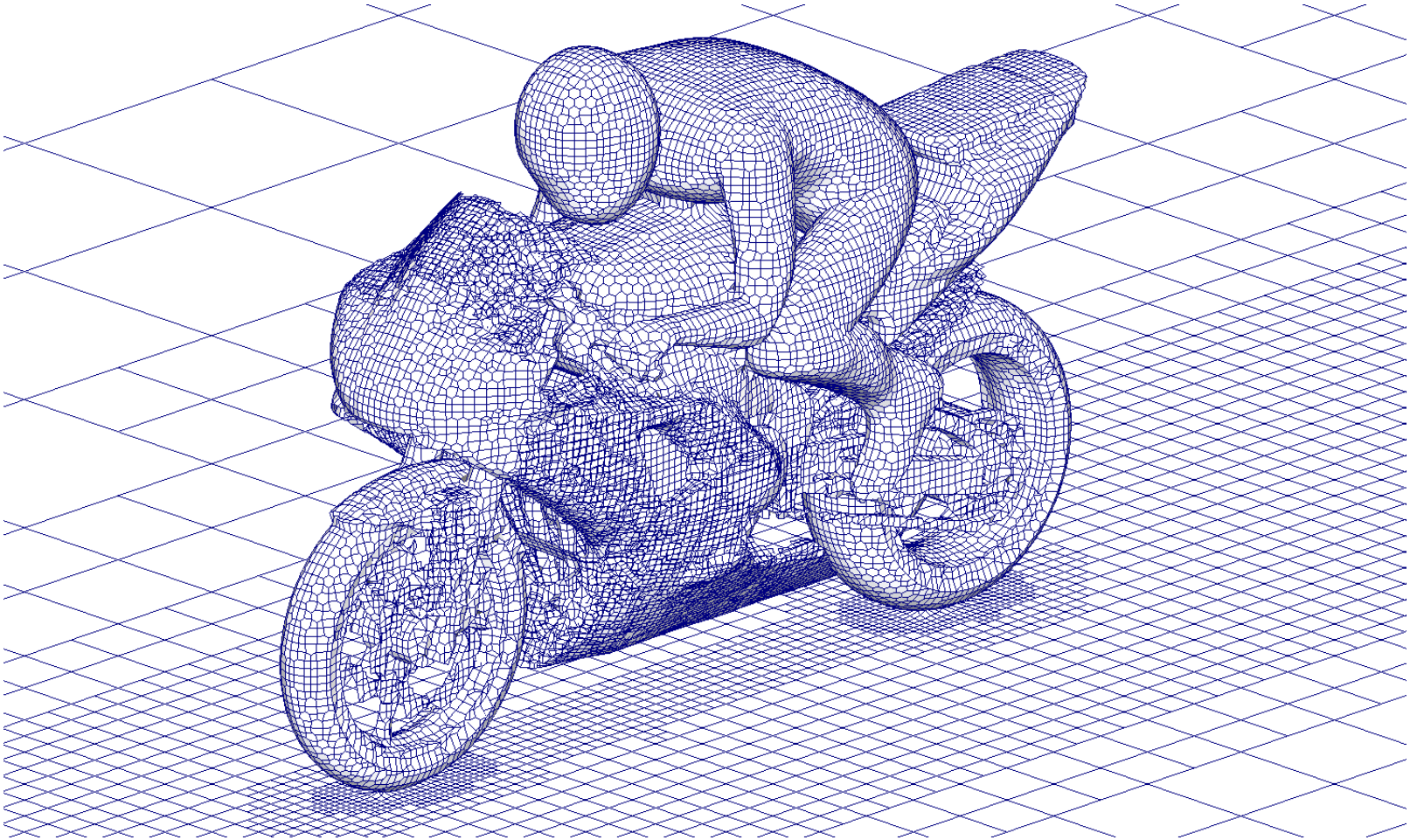
```
<< reconstructParMesh -time 1
<< checkMesh
<< tree
```

Castellated mesh



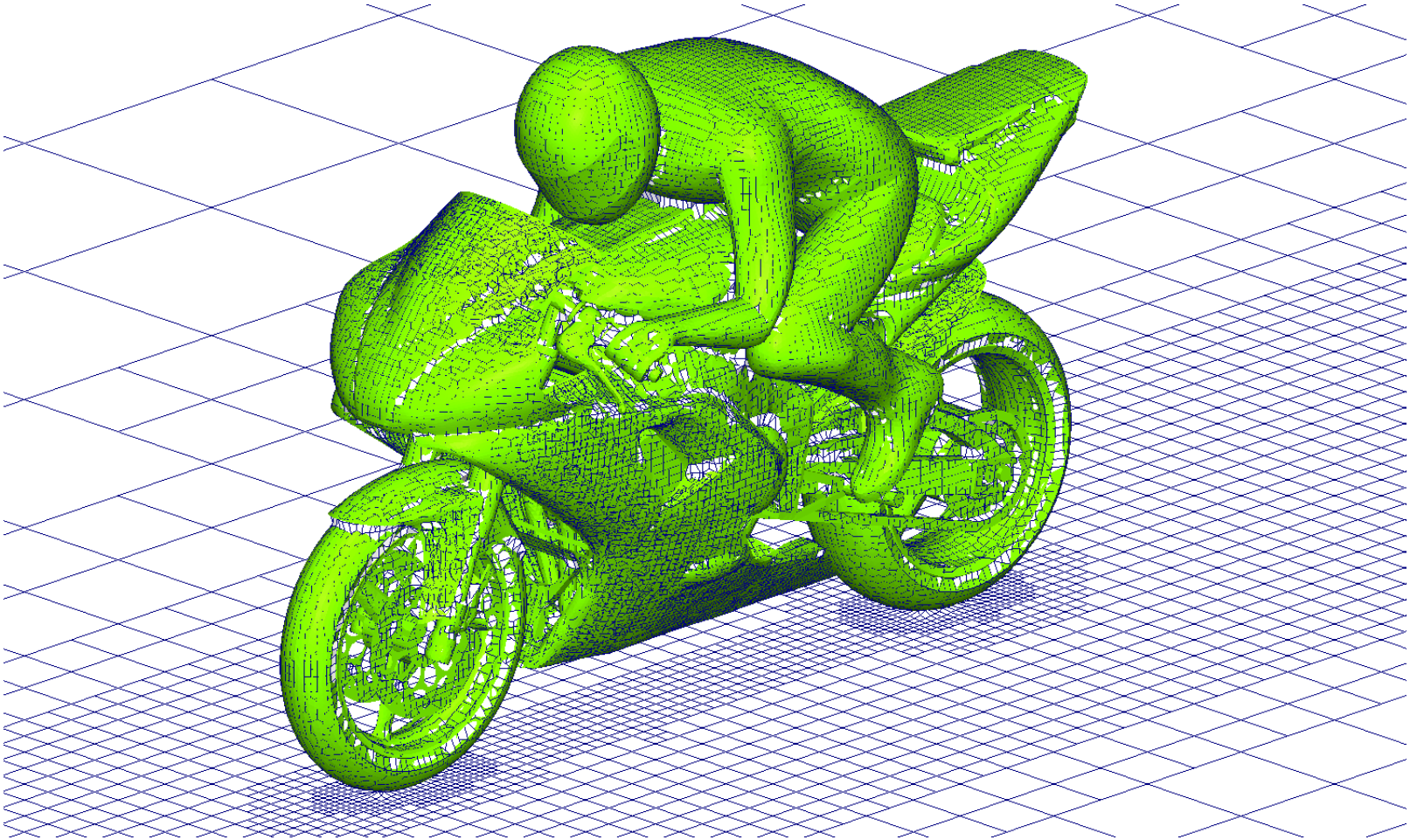
Snapping

Cell faces are projected down onto the surface geometry.



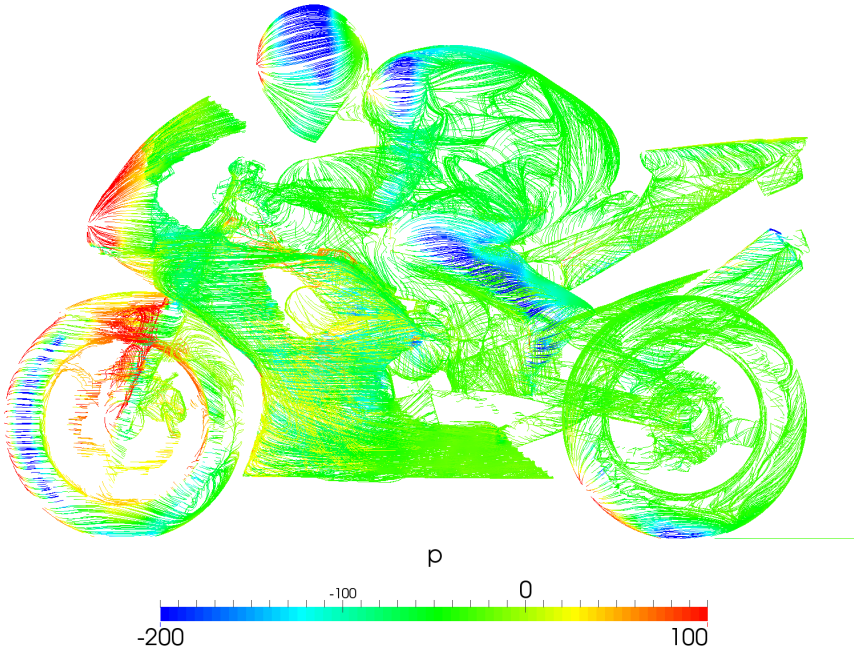
Snapping

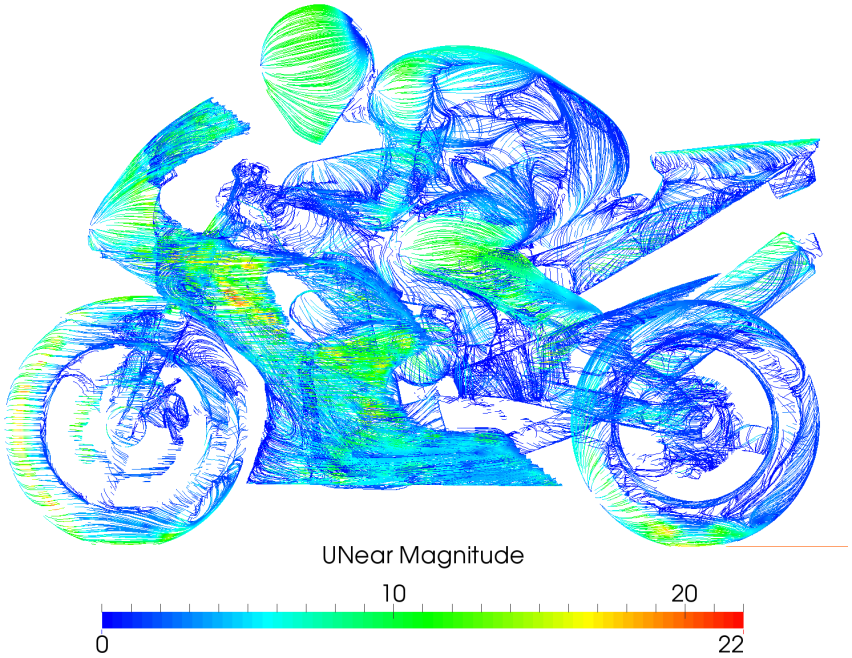
Check the quality of snapping process.



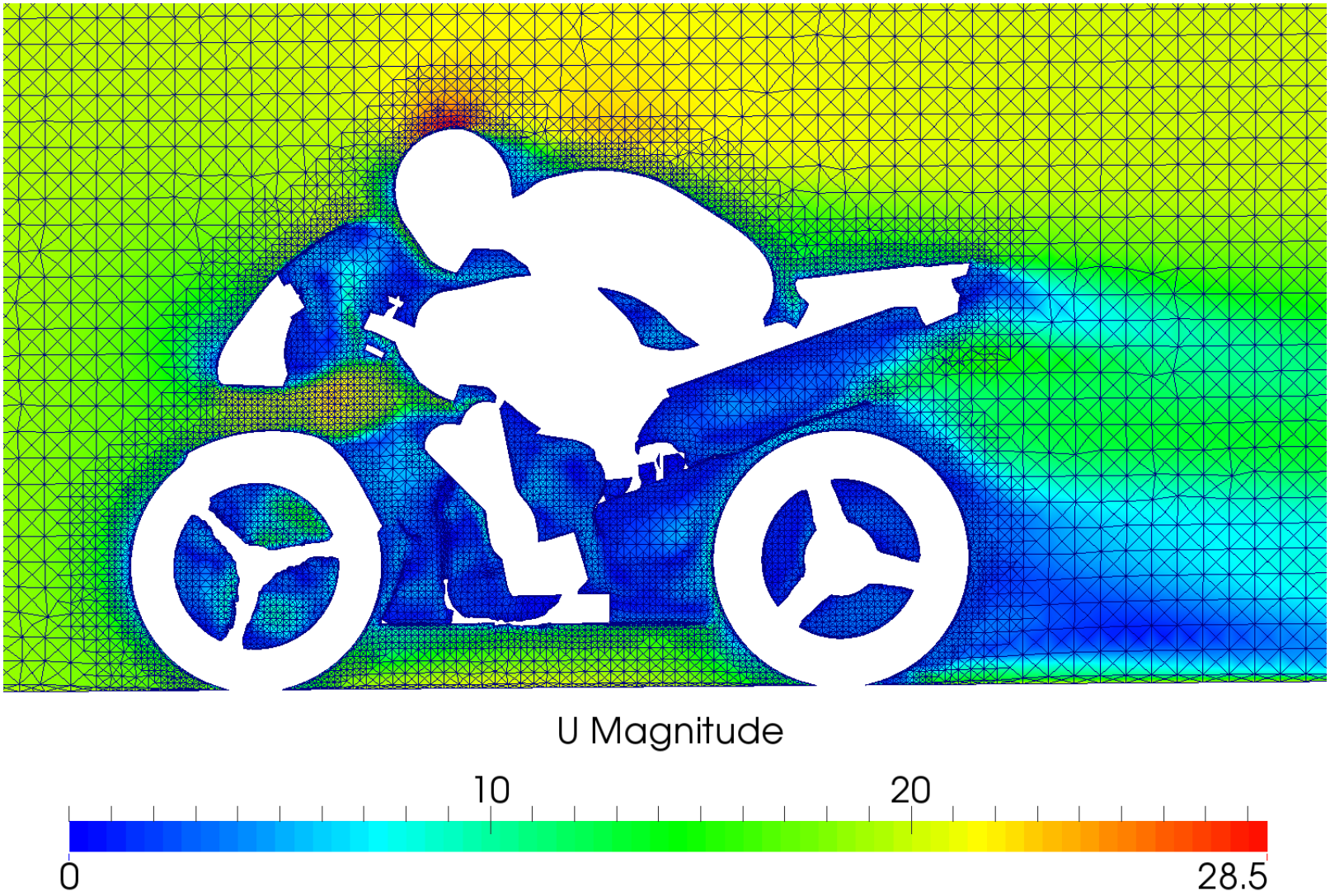
Running the case

- Clean the case: << `./Allclean`
- Run the case: << `./Allrun`





Running the case



How to run OpenFOAM on clonetroop?

Login to clonetroop:

```
<< ssh -X <username>@clonetroop.upc.es
```

Create directory (only for first time usage):

```
<< mkdir -p  /OpenFOAM/OpenFOAM/2.3.1
```

Load **OpenFOAM** environment every time you use it.

```
<< module load apps/openfoam/2.3.1; source $OPENFOAM_SETUP
```

Alternatively for regular **OpenFOAM** use, edit your 'bashrc' file.

Open bashrc file:

```
<< gedit ~/.bashrc
```

Add the following at the end of the file:

```
module load apps/openfoam/2.3.1; source $OPENFOAM_SETUP
```

Scheme of presentation

What have we learnt in his workshop?

Commands:

- ✓ Effective usage of bash shell commands.
- ✓ Navigate through important directories by using environment variables and aliases.

Basics of OpenFOAM:

- ✓ Understand the organisation of **OpenFOAM** installation directory structure.
- ✓ Understand the directory structure of **OpenFOAM** simulation cases.
- ✓ A brief introduction to programming in **OpenFOAM** .

What have we learnt in his workshop?

Meshing:

- ✓ Use *blockMesh* utility to generate simple meshes.
- ✓ Use *snappyHexMesh* utility for generating complex meshes.
- ✓ Use mesh conversion utilities in **OpenFOAM** to use meshes from external softwares.

Solving:

- ✓ Use **OpenFOAM solvers** to run simulations.
- ✓ Use various **OpenFOAM** utilities for parallel simulations.

Visualising:

- ✓ Use **ParaView** to post-process results.

APPENDIX

Useful links

OpenFOAM cheat sheets:

- [Source flux cheat sheet]

OpenFOAM user community:

- [CFD online forum for OpenFOAM]
- [OpenFOAM extend project]

Further information:

- [OpenFOAM wiki page]

Free OpenFOAM course material:

- [PhD course by Prof. Hakan Nilsson at Chalmers]
- [Course by Dr. Joel Guerrero]

Basic Linux commands

Directories:

	Command	Description
1.	`cd <directory>`	Change to a directory location.
2.	`mkdir <directory>`	Create a directory.
3.	`rm -r <directory>`	Remove directory with all its contents.
4.	`cd ..`	Go one level up from current directory.
5.	`cd ../..`	Go two levels up from current directory.
6.	`cd`	Change to the *home* directory.
7.	`pwd`	Print the present working directory path.
8.	`ls <directory>`	List contents of a directory.
9.	`tree`	Print the current directory structure.

Basic Linux commands

Files / Directories:

	Command	Description
1.	`cp <directory1> <directory2>`	Copy a directory from directory path 1 to 2.
2.	`mv <file1> <file2>`	Rename file1 to file2.
3.	`less <file>`	Open a file, press 'q' to quit.
4.	`cat <file>`	Print contents of the file..
5.	`head -10 <file>`	Print first 10 lines of a file.
6.	`tail -50 <file>`	Print last 50 lines of a file.

Basic Linux commands

Other commands:

	Command	Description
1.	`clear`	Clears the terminal window.
2.	`ps`	Displays information about current running processes.
3.	`kill <PID>`	Kills a process ID.
4.	`<command> -help`	See help for a command.
5.	`man <command>`	Manual for a command, press 'q' to quit.

Useful Linux commands for OpenFOAM

Commands for searching:

	Command	Description
1.	`blockMesh -help`	See usage of an OpenFOAM command.
2.	`find $FOAM_TUTORIALS -name sampleDict`	Search *sampleDict* file in '$FOAM_TUTORIALS' directory.
3.	`find $FOAM_SRC -type f | xargs grep -l "::findNearestCell"`	Search the definition of 'findNearestCell' function in '$FOAM_SRC' directory.
4.	`find $FOAM_SRC -type l | xargs grep -l typedef | xargs grep -l "volScalarField;"`	Find the *typedef*, '*volScalarField*' in '$FOAM_SRC' directory.
5.	`find $FOAM_SOLVERS -name "*FvPatch*"`	Find solver specific boundary conditions in '$FOAM_SOLVERS' directory.
6.	`find $FOAM_SOLVERS -name *.C | xargs grep -l incompressible | xargs grep -l multiphase`	Find '*.C' files in '$FOAM_SOLVERS' directory which contain the words 'incompressible' and 'multiphase'.