

CFD on HPC – OpenFOAM

02 – Basics and Meshing



Aleksander GRM – 2025



Cases review

OpenFOAM mesh

Mesh elements

BlockMesh

Introduction

BlockMesh – Examples

GMSH introduction

Introduction

GMSH – Examples

SnappyHexMesh

Basic description

Example

Turbulent Flow Modelling

Why do we need turbulent models?

Turbulence modelling

Turbulence models in OpenFOAM

Show case in OpenFOAM

Cases review



- ▶ BlockMesh introduction
- ▶ GMSH introduction
- ▶ Advanced `cavity_with_hole` show case study
 - ▶ **BlockMesh** mesh: steady and transient case (turbulent $k-\omega$ SST models)
 - ▶ **GMSH** mesh: steady and transient case (turbulent $k-\omega$ SST models)
 - ▶ **SnappyHexMesh** mesh: steady and transient case (turbulent $k-\omega$ SST models)

OpenFOAM mesh



By default OpenFOAM defines a mesh of **arbitrary polyhedral cells** in **3D**, bounded by **arbitrary polygonal faces**, i.e. the cells can have an unlimited number of faces where, for each face, there is no limit on the number of edges nor any restriction on its alignment. A mesh with this general structure is known in OpenFOAM as a **polyMesh**. This type of mesh offers great freedom in mesh generation and manipulation in particular when the geometry of the domain is complex or changes over time.

Basic mesh elements

- ▶ Point
- ▶ Face
- ▶ Cell
- ▶ Boundary

(Detailed mesh description @ cfd.direct)



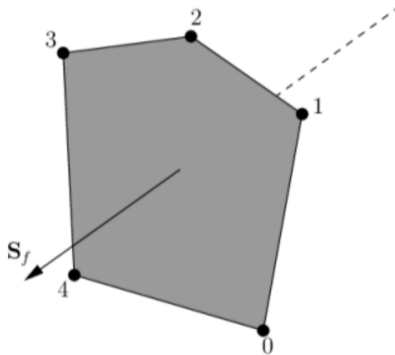
A point is a location in 3-D space, defined by a vector in units of metres (m). The points are compiled into a list and each point is referred to by a label, which represents its position in the list, starting from zero.

The point list cannot contain two different points at an exactly identical position nor any point that is not part at least one face.



```
1 FoamFile
2 {
3     format      ascii;
4     class       vectorField;
5     object      points;
6 }
7 // * * * * *
8
9
10 21812
11 (
12 (-17.5492 0.306481 0)
13 (-17.5472 0.397851 0)
14 (-17.5391 0.49775 0)
15 (-17.5189 0.60624 0)
16 (-17.4861 0.723342 0)
17 ...
18 )
```

- ▶ face is an ordered list of points
- ▶ each two neighbouring points are connected by an edge
- ▶ direction of the face normal vector is defined by the right-hand rule
- ▶ there are two types of face
 - ▶ Internal faces
 - ▶ Boundary faces





```
1 FoamFile
2 {
3     format      ascii;
4     class       faceList;
5     object      faces;
6 }
7 // * * * * *
8
9 43066
10 (
11 4(156 0 78 235)
12 4(157 236 79 1)
13 4(156 235 236 157)
14 4(158 237 80 2)
15 ...
16 )
```



A cell is a list of faces in arbitrary order. Cells must have the properties listed below.

- ▶ The cells must be **contiguous**, i.e. completely cover the computational domain and must not overlap one another.
- ▶ Every cell must be **closed geometrically**, such that when all face area vectors are oriented to point outwards of the cell, their sum should equal the zero vector to machine accuracy;
- ▶ Every cell must be **closed topologically** such that all the edges in a cell are used by exactly two faces of the cell in question.



```
1 FoamFile
2 {
3     format      ascii;
4     class       cellList;
5     object      cells;
6 }
7 // * * * * *
8
9 10720
10 (
11 6(0 1 21626 21627 2 21548)
12 6(1 3 21628 21629 4 21549)
13 6(3 5 21630 21631 6 21550)
14 ...
15 )
```



- ▶ **points:** a list of vectors describing the cell vertices, where the first vector in the list represents vertex 0, the second vector represents vertex 1, etc. ;
- ▶ **faces:** a list of faces, each face being a list of indices to vertices in the points list, where again, the first entry in the list represents face 0, etc. ;
- ▶ **owner:** a list of owner cell labels, the index of entry relating directly to the index of the face, so that the first entry in the list is the owner label for face 0, the second entry is the owner label for face 1, etc;
- ▶ **neighbour:** a list of neighbour cell labels;
- ▶ **boundary:** a list of patches, containing a dictionary entry for each patch, declared using the patch name



We will follow the text in `cfD.direkt` web page, explaining:

- ▶ geometry boundaries
- ▶ different types of standard boundary conditions

OpenFOAM v10 User Guide: 5.2 Boundaries

BlockMesh



The **blockMesh** mesh generator is generator of **structured mesh**. In most of the cases this is most desired type of mesh.

- ▶ The blockMesh utility creates parametric meshes with grading and curved edges.
- ▶ The mesh is generated from a dictionary file named blockMeshDict located in the system (or constant/polyMesh) directory of a case. blockMesh reads this dictionary, generates the mesh and writes out the mesh data to points and faces, cells and boundary files in the same directory.
- ▶ The principle behind blockMesh is to decompose the domain geometry into a set of 1 or more three dimensional, hexahedral blocks. Edges of the blocks can be straight lines, arcs or splines. The mesh is ostensibly specified as a number of cells in each direction of the block, sufficient information for blockMesh to generate the mesh data.



We shall continue **BlockMesh** learning over the web UserGuide @ OpenFoam.org

5.4 BlockMesh @ cfd.direct



Folder: day-02/01_blockMesh

- ▶ show the **cavity** example.
- ▶ show the **cavity_with_hole** example.

GMSH introduction



To be able to run advanced GMSH examples we need to set up Python environment

```
1 1. load module python:
2   $> ml av python (check target version)
3   $> ml python-version
4
5 2. Create new env:
6   $> python3 -m venv local
7
8 3. Activate new env:
9   $> source local/bin/activate
10
11 4. Install new packages (active env local):
12  $(local)> pip install numpy scipy sympy matplotlib gmsh
```



To use Python environment we need only to load it

```
1 1. load module python:
2   $> ml av python (check target version)
3   $> ml python-version
4
5 2. Activate new env:
6   $> source local/bin/activate
```



We shall continue **GMSH** description/learning over the web UserGuide @ gmsh.info

User Guide – vLatest@ gmsh.info



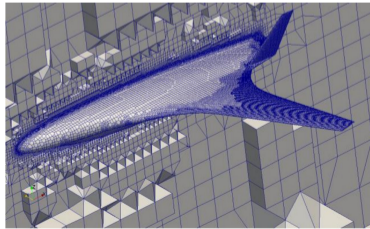
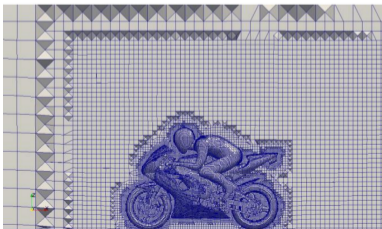
Folder: day-02/02_GMSH

- ▶ show the **cavity** example.
- ▶ show the **cavity_with_hole** example.

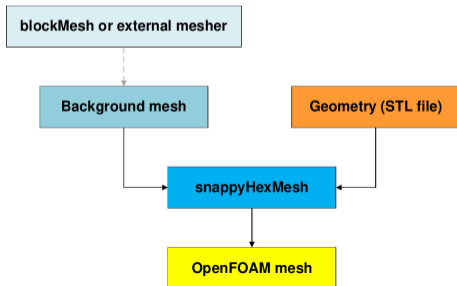
Emphasise how GMSH mesh is converted to OpenFOAM mesh!

SnappyHexMesh

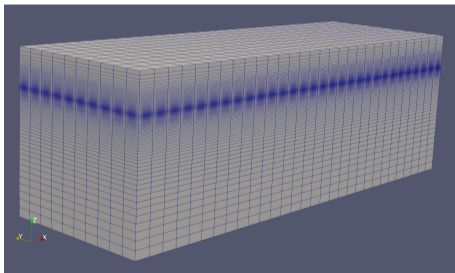
- ▶ *Automatic split hex mesher. Refines and snaps to surface.*
- ▶ The **snappyHexMesh** utility generates 3D meshes containing hexahedra and split-hexahedra from a triangulated surface geometry in Stereolithography (STL) or Wavefront (OBJ) format.
- ▶ The mesh is generated from a dictionary file named `snappyHexMeshDict` located in the system directory and a triangulated surface geometry file located in the directory `constant/triSurface` or `constant/geometry`.



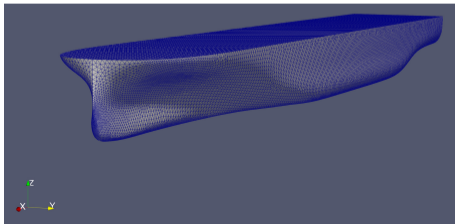
- ▶ Generation of a background or base mesh
- ▶ Geometry definition
- ▶ Generation of a castellated mesh or Cartesian mesh
- ▶ Generation of a snapped mesh or body fitted mesh
- ▶ Addition of layers close to the surfaces or boundary layer meshing
- ▶ Check/enforce mesh quality



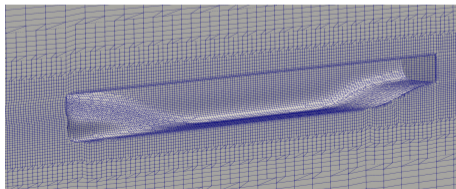
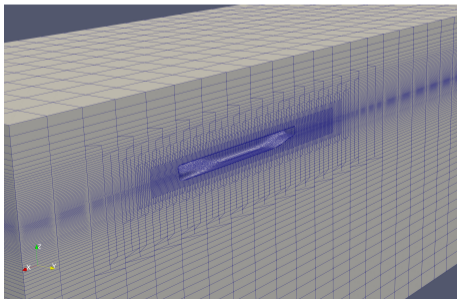
- ▶ The background or base mesh can be generated using blockMesh or an external mesher
- ▶ The following criteria must be observed when creating the background mesh
 - ▶ The mesh must consist purely of hexes
 - ▶ The cell aspect ratio should be approximately 1, at least near the STL surface
 - ▶ There must be at least one intersection of a cell edge with the STL surface



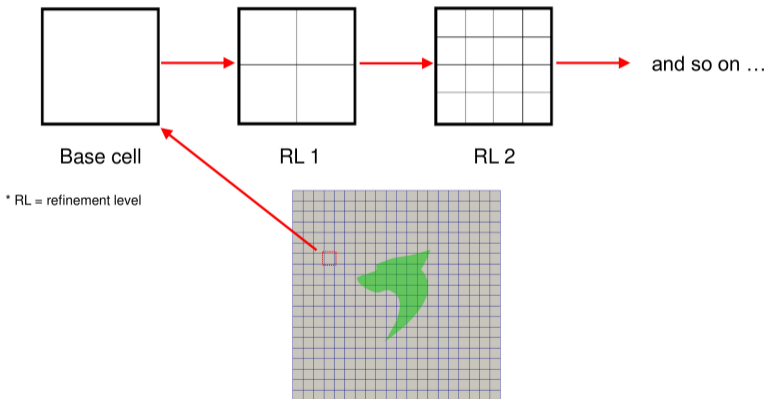
- ▶ is imported from STL/OBJ file
- ▶ can be made up of a single surface describing the geometry, or multiple surfaces that describe the geometry
- ▶ it has a freedom not to be watertight
- ▶ geometry with multiple surfaces, we can use local refinement in each individual surface.
- ▶ geometry is always located in the directory
`constant/geometry` or `constant/triSurface`



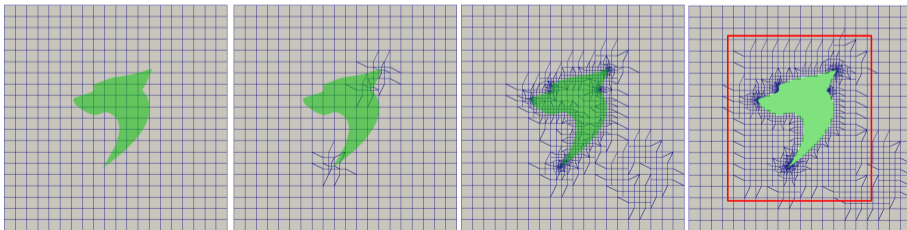
- ▶ The meshing utility **snappyHexMesh** reads the dictionary `system/snappyHexMeshDict`
- ▶ The **castellation**, **snapping**, and **boundary layer** meshing steps are controlled by the dictionary `snappyHexMeshDict`
- ▶ The final mesh is always written in the default OpenFOAM mesh directory `constant/polyMesh` and ready to use for simulation



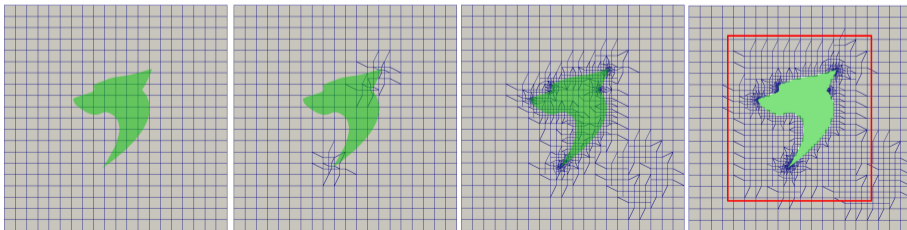
- Mesh refinement is based on the background mesh!



- ▶ creating background hexahedral mesh
- ▶ refine mesh at featured edges
- ▶ refine mesh at surfaces
- ▶ cell removal
- ▶ snapping mesh cells to surface
- ▶ creating boundary layer mesh if needed

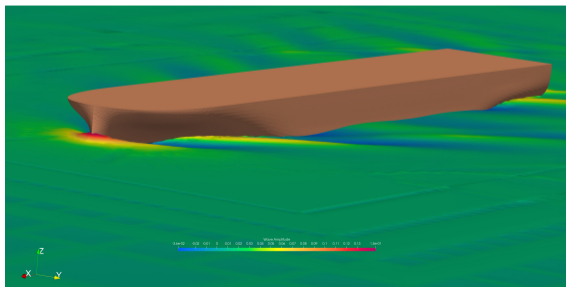


- ▶ creating background hexahedral mesh
- ▶ refine mesh at featured edges
- ▶ refine mesh at surfaces
- ▶ cell removal
- ▶ snapping mesh cells to surface
- ▶ creating boundary layer mesh if needed



To generate a mesh for ship resistance analysis, three dictionary files must be written

- ▶ **generate background mesh:** `system/blockMeshDict`
- ▶ **detect featured edges:** `system/surfaceFeaturesDict`
- ▶ **generate internal mesh:** `system/snappyHexMeshDict`





- ▶ **run command:** blockMesh
- ▶ open file
- ▶ comment parameters
- ▶ comment vertices and blocks
- ▶ comment boundary faces



- ▶ **run command:** `surfaceFeaturesExtract`
- ▶ open dictionary file
- ▶ comment on surfaces import:

```
surfaces ("hull.obj");  
surfaces ("hull.obj" "deck.obj");
```
- ▶ comment `includedAngle`: Mark edges whose adjacent surface normals are at an angle less than `includedAngle` as features
- ▶ comment `subsetFeatures`
- ▶ comment `trimFeatures`

[link to surfaceFeatureExtract.C](#)



- ▶ **run command:** `snappyHexMesh`
- ▶ open dictionary file
- ▶ Which of the steps to run
 - `castellatedMesh` `true/false;`
 - `snap` `true/false;`
 - `addLayers` `true/false;`
- ▶ comment `geometry`
- ▶ comment `castellatedMeshControls`
- ▶ comment `snapControls`
- ▶ comment `addLayersControls`

Turbulent Flow Modelling



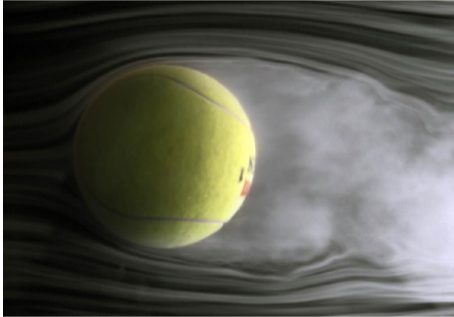
Most of engineering CFD problems are in **turbulent regime**!

- ▶ **external flow**: $Re > 500,000$ (slender bodies), $Re > 20,000$ (obstacles)
- ▶ **internal flow**: $Re > 2,300$

Reynolds number Re

$$Re = \frac{U L}{\nu}, \quad \nu = \frac{\mu}{\rho}$$

- ▶ U : problem undisturbed flow velocity magnitude,
- ▶ L : problem length scale,
- ▶ μ : dynamic viscosity,
- ▶ ρ : density
- ▶ ν : kinematic viscosity.



Wind tunnel test of new tennis ball.

(This we will try to solve in 2D!)



Wake turbulence behind individual wind turbines



"Turbulence is the most important unresolved problem of classical physics"

Richard Feynman



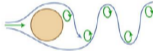



"Turbulence was probably invented by the devil on the seventh day of creation when the good lord was not looking"

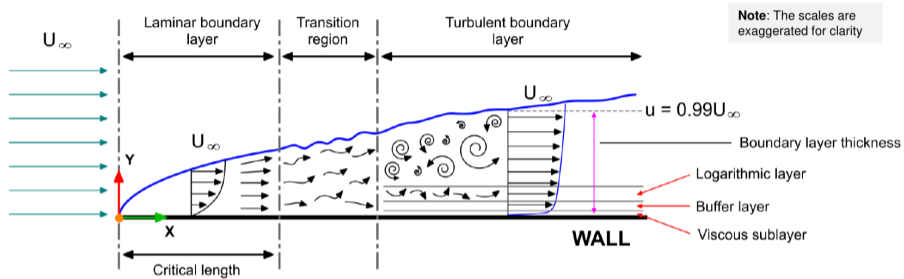
Peter Bradshaw

"Turbulence is the graveyard of theories"

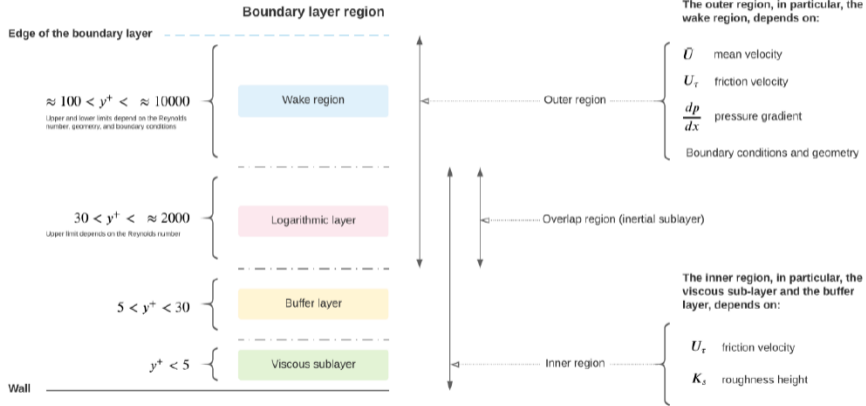
Hans W. Liepmann

- ▶ Most natural and engineering flows are turbulent, hence the necessity of modelling turbulence.
- ▶ The goal of turbulence modelling is to develop equations that predict the **time averaged** velocity, pressure, temperature fields without calculating the complete turbulent flow pattern as a function of time.
- ▶ Turbulence can be wall bounded or free shear. Depending on what you want to simulate, you will need to choose an appropriate turbulence model.
- ▶ There is no universal turbulence model, hence you need to know the capabilities and limitations of the turbulence models.
- ▶ Due to the multi-scale and unsteady nature of turbulence, modelling it is not an easy task.
- ▶ Simulating turbulent flows in any general CFD solver (e.g., OpenFOAM, SU2, Fluent, CFX, Star-CCM+) requires selecting a turbulence model, providing initial conditions and boundary conditions for the closure equations of the turbulent model, selecting a near-wall modelling treatment, and choosing runtime parameters and numerics.

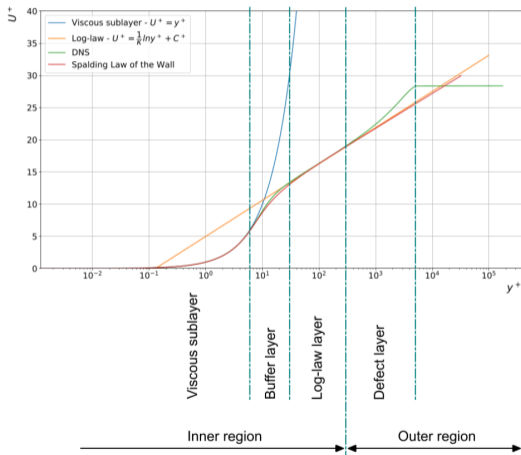
| | | | |
|---|---|--|---|
|  | Creeping flow (no separation) Steady flow | $Re < 5$ | <ul style="list-style-type: none"> • Easy to simulate • Steady |
|  | A pair of stable vortices in the wake Steady flow | $5 < Re < 40 - 46$ | |
|  | Laminar vortex street (Von Karman street) Unsteady flow | $40 - 46 < Re < 150$ | <ul style="list-style-type: none"> • Relatively easy to simulate. • It becomes more challenging when the boundary layer transition to turbulent • Unsteady |
|  | Laminar boundary layer up to the separation point, turbulent wake Unsteady flow | $150 < Re < 300$ Transition to turbulence $300 < Re < 3 \times 10^5$ | |
|  | Boundary layer transition to turbulent Unsteady flow | $3 \times 10^5 < Re < 3 \times 10^6$ | |
|  | Turbulent vortex street, but the wake is narrower than in the laminar case Unsteady flow | $3 \times 10^6 > Re$ | <ul style="list-style-type: none"> • Challenging to simulate • Unsteady |



- ▶ a laminar boundary layer starts to form at the leading edge.
- ▶ as the flow proceeds further downstream, large shear stresses and velocity gradient develop within the boundary layer. At one point the flow becomes turbulent.
- ▶ the turbulent motion increases the mixing and the boundary layer mixing.
- ▶ what is happening in the transition region is not well understood. The flow can become laminar again or can become turbulent.
- ▶ as for the pipe flow, the velocity profiles in the laminar and turbulent regions are different.



The boundary layer explanation!



Law of the wall
(**LoW**)

$$u^+ = f(y^+)$$

Non-dimensional
velocity

$$u^+ = \frac{u}{u_\tau}$$

Non-dimensional y
scale

$$y^+ = \frac{y u_\tau}{\nu}$$

LoW is the cornerstone law for the boundary layer mesh design!



► Viscous sublayer

- The viscous sublayer, refers to the region of the inner-region of the boundary layer, very close to the wall and where the flow is *laminar*.
- In this region the flow mean velocity can be described using a simple analytic function.
- The viscous sublayer law, is stated as follows

$$u^+ = y^+$$

- Remember, this equation is only valid in the viscous sublayer, where the flow is laminar and viscous effect are very strong, therefore,

$$\tau_{\text{wall}} = \mu \frac{\partial u}{\partial y}$$

- According to this law, the behaviour of the mean velocity is linear in this region.



► Logarithmic law or log-law

- The logarithmic law, refers to the region of the inner-region of the boundary layer that can be described using a simple analytic function in the form of a logarithmic equation.
- This is one of the most famous empirically determined relationships in turbulent flows *near solid boundaries*.
- Measurements show that, for both **internal** and **external** flows, the stream-wise velocity in the flow near the wall varies logarithmically with distance from the surface.
- The log-law, is stated as follows,

$$\boxed{u^+ = \frac{1}{\kappa} \log y^+ + C^+}, \quad \text{the most common values } \kappa = 0.41, \quad C^+ = 5.0$$

- Reported values for the constant C^+ can go anywhere from 4.5 to 5.5.
- Reported values of the Karman constant κ can go anywhere from 0.36 to 0.42.



- ▶ **Viscous sublayer** – The viscous sublayer law is valid for values of y^+ ranging from,

$$y^+ < 5$$

- ▶ **Buffer layer** – The buffer layer is enclosed in the following range of y^+ values,

$$5 < y^+ < 30$$

- ▶ **Logarithmic law layer**

- ▶ The logarithmic law or log-law is valid for values of y^+ ranging from,

$$30 < y^+ < 300$$

- ▶ For practical purposes in CFD

$$30 < y^+ < 600$$



► Needed parameters

to find first near wall cell height

- U_∞ : free stream velocity in [m/s]
- L : problem length scale in [m]
- ρ : fluid density [kg/m³]
- μ : fluid dynamic viscosity [Pa s]
- y^+ : layer resolution parameter

► Calculation steps

- Kinematic viscosity [m²/s]

$$\nu = \frac{\mu}{\rho}$$

- Reynolds number

$$Re = \frac{U_\infty L}{\nu}$$

- Friction coefficient

$$C_f = \left(2 \log_{10}(Re) - 0.65\right)^{-2.3}$$

- Wall shear stress

$$\tau_w = C_f \frac{1}{2} \rho U_\infty^2$$

- Friction velocity

$$U_\tau = \sqrt{\frac{\tau_w}{\rho}}$$

- Near wall layer thickness

$$y = h_1 = \frac{y^+}{U_\tau} \frac{\mu}{\rho} = \frac{y^+}{U_\tau} \nu$$



► Viscous sublayer

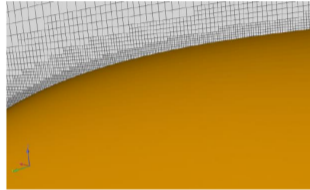
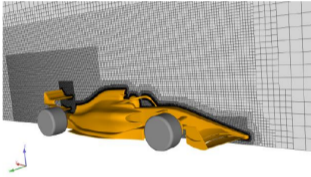
- if there is flow separation
- if using laminar flow model (need to resolve viscous sublayer)
- first near wall cell height h_1 is based on $y^+ = 1$
- needed 5-10 layers with inflation scaling factor $\alpha = 1.2$

► Logarithmic law layer

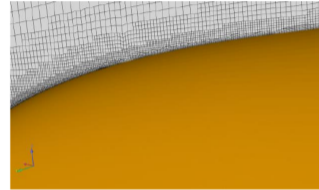
- *not needed to resolve viscous sublayer*
- use of *wall function* in turbulent model ($k-\omega$ SST)
- first near wall cell height h_1 is based on $y^+ \approx 30$
- needed N layers with scaling factor $\alpha = 1.2$
- N can be computed using boundary layer thickness information δ

$$\delta = H = 0.37 L Re^{-1/5}, \quad N = 1 + \frac{\log \left(1 - \frac{H}{h_1} (1 - \alpha) \right)}{\log \alpha}$$

where $\alpha = 1.2$ is inflation factor, h_1 thickness of the first near wall layer, found on the previous slide.

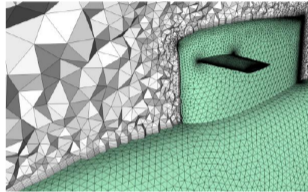
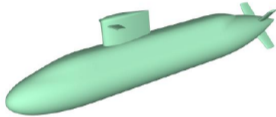


Wall modeling mesh
Average y^+ approximately 60

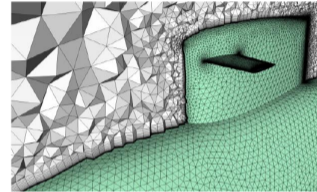


Wall resolving mesh
Average y^+ approximately 1

| | Wall modeling mesh | Wall resolving mesh |
|-----------------|--------------------|---------------------|
| Number of cells | 57 853 037 | 111 137 673 |



Wall modeling mesh
Average y^+ approximately 60



Wall resolving mesh
Average y^+ approximately 1

| | Wall modeling mesh | Wall resolving mesh |
|-----------------|--------------------|---------------------|
| Number of cells | 6 613 049 | 11 149 266 |



Folder: tools and literature/mesh calculator

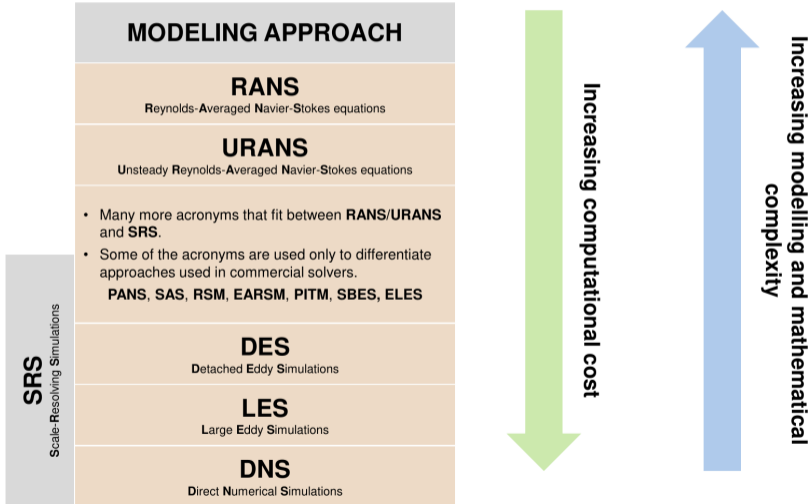
► `mesh_size_calculator.ipynb`

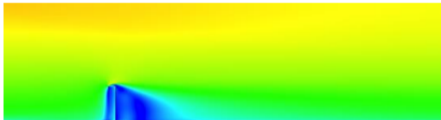
→ calculates the boundary layer mesh parameters.

► `k_omega_initial_conditions.ipynb`

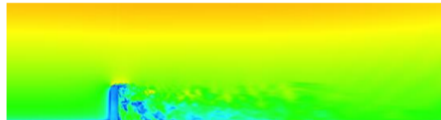
→ calculates the initial values for k and ω OpenFOAM fields.

Files `*.ipynb` are interactive python notebooks, used in **Jupyter** (jupyter.org)





RANS



LES – Instantaneous field

| RANS/URANS | DES/LES | DNS |
|---|--|--|
| <ul style="list-style-type: none"> Solve the time-average NSE. All turbulent spatial scales are modeled. Many models are available. One equation models, two equation models, Reynolds stress models, transition models, and so on. This is the most widely approach for industrial flows. Unsteady RANS (URANS), use the same equations as the RANS but with the transient term retained. It can be used in 2D and 3D cases. | <ul style="list-style-type: none"> Solve the filtered unsteady NSE. Sub-grid scales (SGS) are filtered, grid scales (GS) are resolved. Aim at resolving the temporal scales, hence requires small time-steps. For most industrial applications, it is computational expensive. However, thanks to the current advances in parallel and scientific computing it is becoming affordable. Many models are available. It is intrinsically 3D and asymmetric. | <ul style="list-style-type: none"> Solves the unsteady NSE with no models involved (laminar NSE). Solves all spatial and temporal scales; hence, requires extremely fine meshes and small time-steps. No modeling is required. It is extremely computational expensive. Not practical for industrial flows. It is intrinsically 3D and asymmetric. |



- **RANS**: used to model *steady* turbulent flow

→ $k - \omega$ SST model (in **OF** under RAS – kOmegaSST)

- **URANS**: used to model *transient* turbulent flow

→ $k - \omega$ SST-SAS model (in **OF** under RAS – kOmegaSSTSAS)

$k - \omega$ SST model is considered to be the most **reliable** turbulent model in RANS family!

RANS and **URANS** family of turbulent models do time window solution averaging!
The job is to smear out vortices and hide the vortex energy in turbulent viscosity ν_T .

File: constant/transportProperties

```
1
2 transportModel    Newtonian;
3
4 // *** Air @ 20 celsius ***
5 //rho             [1 -3 0 0 0 0 0] 1.2;      // density
6 //nu              [0 2 -1 0 0 0 0] 1.5e-05; // kinematic viscosity
7
8 // *** Water @ 20 celsius ***
9 rho              [1 -3 0 0 0 0 0] 1000; // density
10 nu              [0 2 -1 0 0 0 0] 1e-06; // kinematic viscosity
```



File: constant/turbulenceProperties

```
1 // *** laminar model ***
2 //simulationType laminar;
3
4 // ** RAS turbulent model ***
5 simulationType RAS;
6
7 RAS
8 {
9     RASModel          kOmegaSST;
10    turbulence         on;
11    printCoeffs        on;
12 }
```



File: constant/turbulenceProperties

```
1 // *** laminar model ***
2 //simulationType laminar;
3
4 // ** RAS turbulent model ***
5 simulationType RAS;
6
7 RAS
8 {
9     RASModel      kOmegaSSTsas;
10    turbulence     on;
11    printCoeffs    on;
12
13    delta          vanDriest;
```

```
15    vanDriestCoeffs
16    {
17        delta          cubeRootVol;
18        cubeRootVolCoeffs
19        {
20            deltaCoeff      1;
21        }
22
23        smoothCoeffs
24        {
25            delta          cubeRootVol;
26            cubeRootVolCoeffs
27            {
28                deltaCoeff      1;
29            }
30
31            maxDeltaRatio    1.1;
32        }
33
34        Aplus              26;
35        Cdelta              0.158;
36    }
37
38 }
```



To realistically model a given problem, it is important to define the turbulence intensity at the inlets. Here are a few examples of common estimations of the incoming turbulence intensity:

- ▶ **High-turbulence:** (between 5% and 20%): Cases with high velocity flow inside complex geometries. Examples: heat exchangers, flow in rotating machinery like fans, engines, etc.
- ▶ **Medium-turbulence** (between 1% and 5%): Flow in not-so-complex geometries or low speed flows. Examples: flow in large pipes, ventilation flows, etc.
- ▶ **Low-turbulence** (well below 1%): Cases with fluids that stand still or highly viscous fluids, very high-quality wind tunnels. Examples: external flow across cars, submarines, aircraft, etc.



Folder: day-02/03_cavity_with_hole

► **Block Mesh** mesh model case:

- steady case: use of $k - \omega$ SST turbulent model
- transient case: (show comparison for $Re = 100, 200, 1000, 10000$)
 - laminar model
 - use of $k - \omega$ SST-SAS turbulent model

► **GMSH** mesh model case:

- steady case: use of $k - \omega$ SST turbulent model
- transient case: (show comparison for $Re = 100, 200, 1000, 10000$)
 - laminar model
 - use of $k - \omega$ SST-SAS turbulent model

► **fluidmechanics101@youtube.com**: link



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.