Disclaimer

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

Advanced OpenFOAM® Training
October 10th, 2016

Naval Applications using OpenFOAM® technology

DITEN

Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture
Polytechnic School, University of Genoa

Copyright and disclaimer

.

Authors makes no warranty, express or implied, about the completeness, accuracy, reliability, suitability, or usefulness of the information disclosed in this training material. This training material is intended to provide general information only. Any reliance the final user place on this training material is therefore strictly at his/her own risk. Under no circumstances and under no legal theory shall authors be liable for any loss, damage or injury, arising directly or indirectly from the use or misuse of the information contained in this training material.

Contact information

Stefano GAGGERO
stefano.gaggero@unige.it

University of Genova
DITEN – Department of Electrical, Electronic and Telecommunications
Engendering and Naval Architecture

Contact information

Diego VILLA

diego.villa@unige.it

University of Genova

DITEN – Department of Electrical, Electronic and Telecommunications Engendering and Naval Architecture

Today's lecture

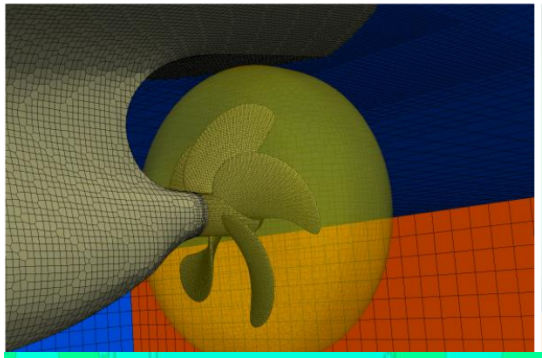
1. Introduction to the hydrodynamic naval problems
    1. Ship Resistance
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

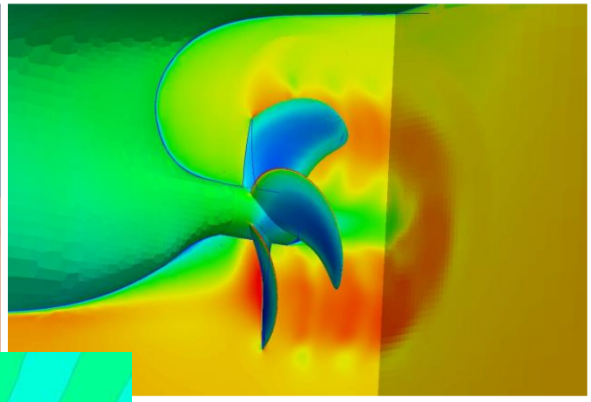
Today's lecture

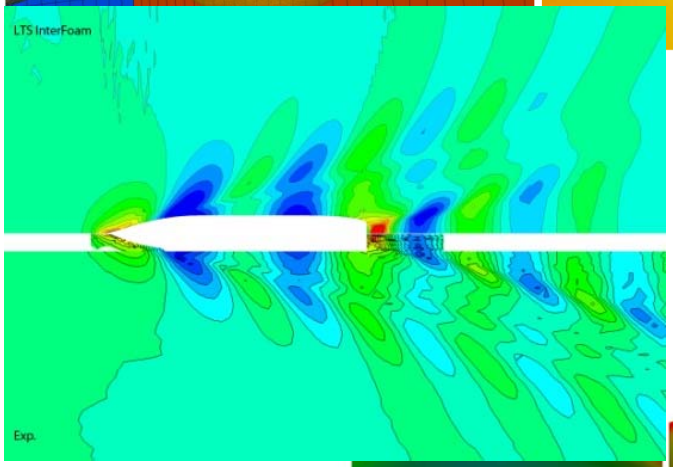
1. **Introduction to the hydrodynamic naval problems**
    1. Ship Resistance
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

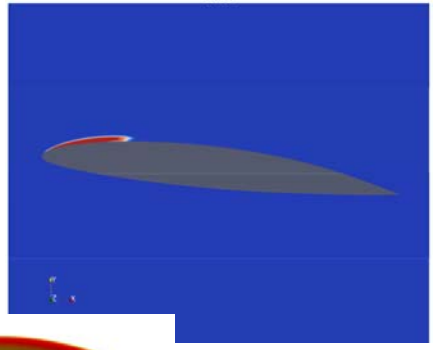
Naval Hydrodynamic Problem

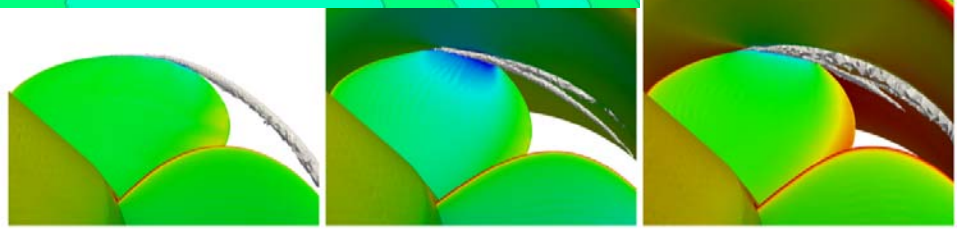
In the Naval field the CFD (Computational Fluid Dynamic) is a strategic key point to predict the performance of a Ship.



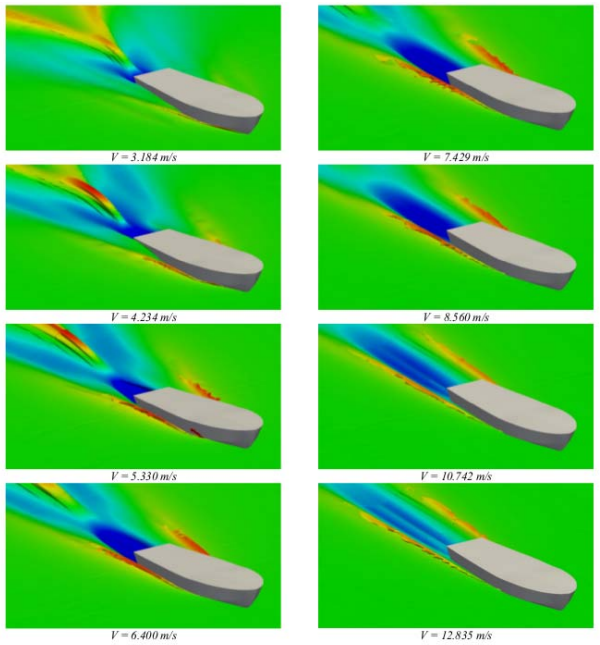


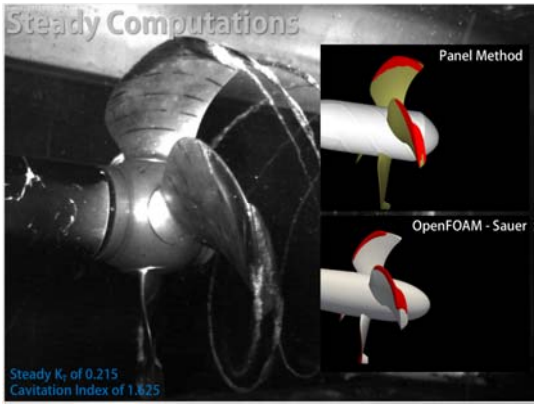






(a) Free running propeller (b) Accelerating Duct (c) Decelerating Duct





Today's lecture

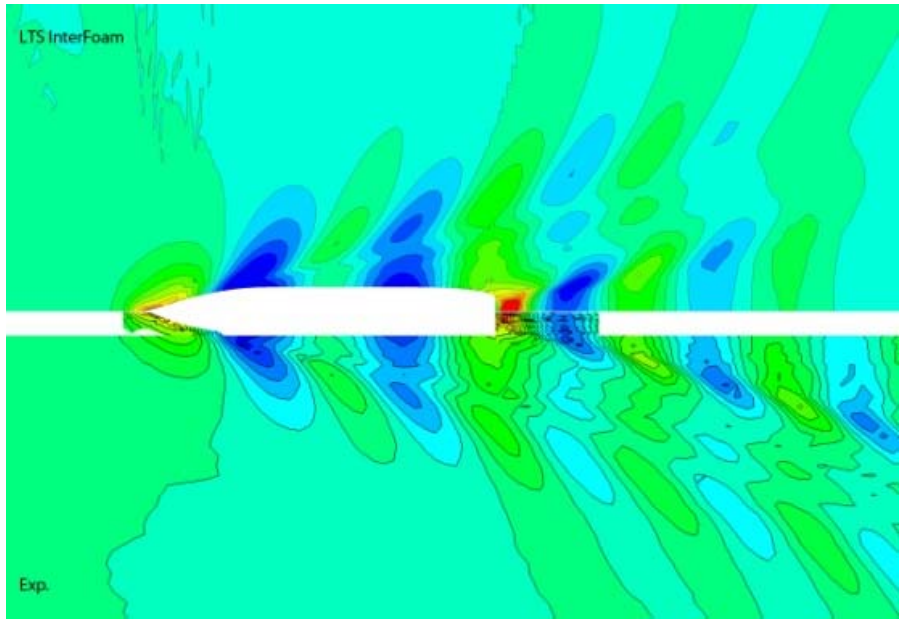
1. **Introduction to the hydrodynamic naval problems**
    1. **Ship Resistance**
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

Ship Resistance

The ship resistance can be split in three main components in accordance with the equation:

$$C_T = C_W + C_F\,(1 + K)$$

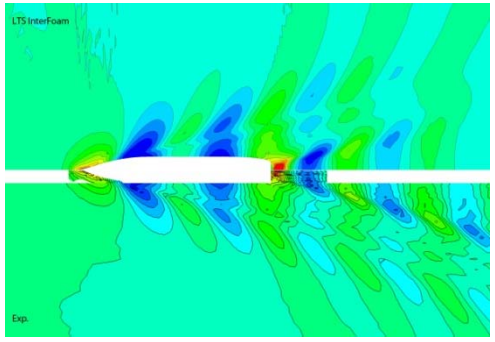
C_T = Total Resistance
C_F = Frictional Resistance
C_W = Wave Resistance
K = Form Factor



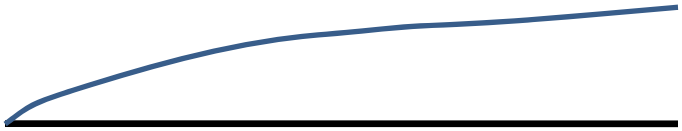
Ship Resistance

Each term represents a different contribution of force, therefore they follow different scale factor:

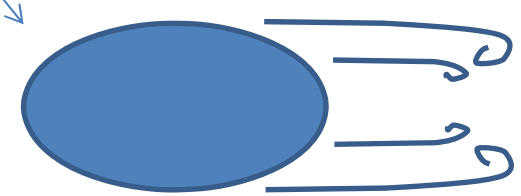
$$C_T = C_W + C_F\,(1 + K)$$



Wave pattern f(Fn)
Energy needed to move the free surface



Equivalent flat plate f(Rn)
Integral of all the shear stress tensions

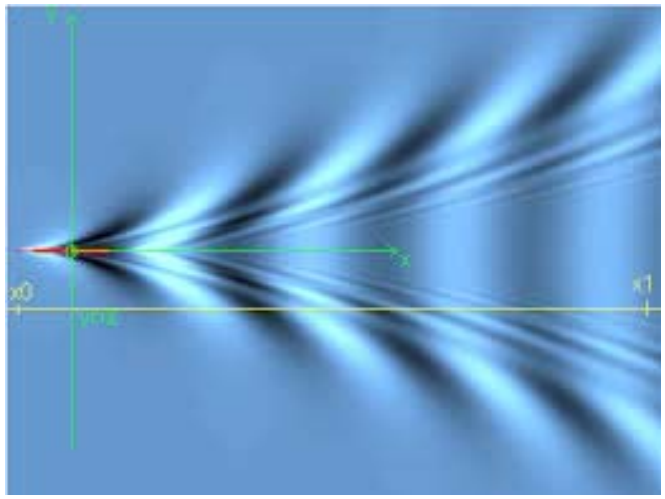


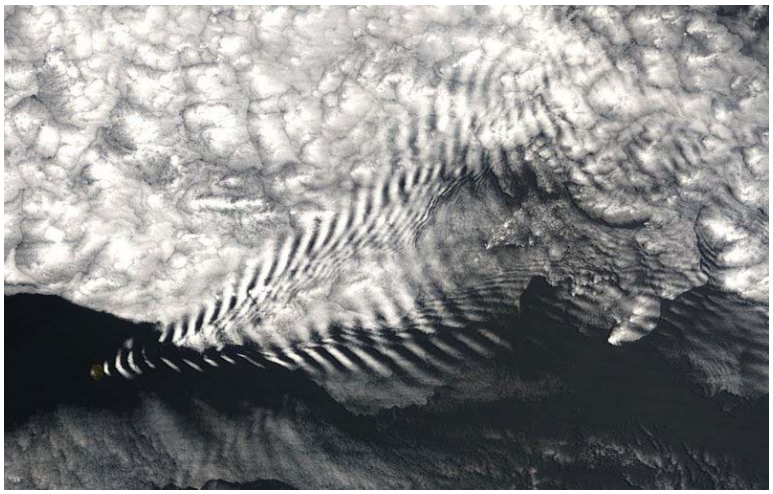
Wake f(Rn,Fn…)
Integral of all the pressure excluding the wave component
Variation of the boundary layer due to the body shape

Ship Resistance

C_W - The wave resistance in the past has been extensively evaluated with Potential Code. This method is based on the hypotheses to neglect the viscosity effects on the governing equations, therefore the Eulerian equation can be used. Moreover adding the hypothesis to have an irrotational motion, the problem can be further simplified writing the problem in term of an harmonic velocity potential.

$$\Delta\varphi = 0$$
$$\nabla\varphi = \vec{V}$$

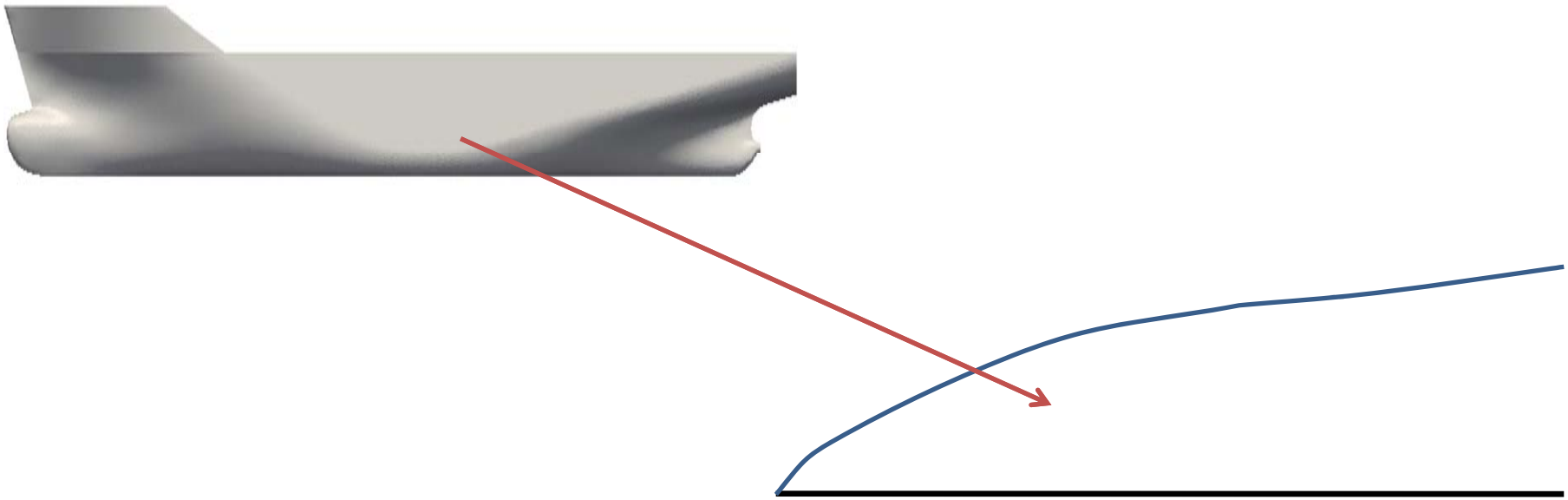




http://taflab.berkeley.edu/ME245-SP14/ME245_Oceanic_Atmospheric_Waves_Examples.htm

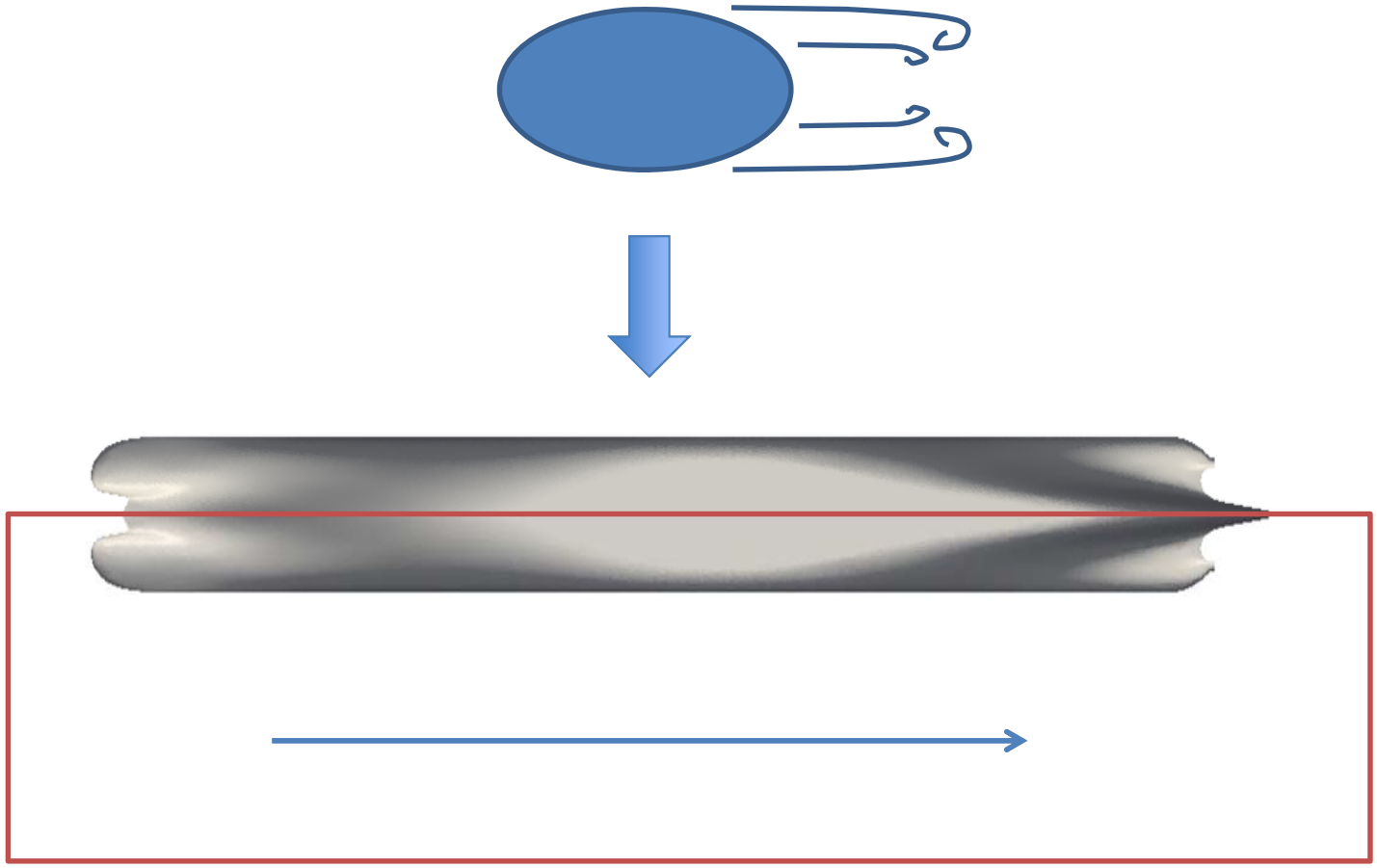
Ship Resistance

C_F – The potential approach is not able to take in to account the wall shear stress (due to the hp. to neglect the viscosity). In the past some empirical formulation have been used like the flat plate equivalence…



Ship Resistance

K – To evaluate the wake effects some ad-hoc experimental tests should be used. Only in last decade using the Double Model approach the viscous are able to give some results.



Ship Resistance

Using a modern Viscous solver the direct comparison with experimental results should be made only between the total forces. The force decomposition can be a good guideline, but more discrepancies can shown.

Ship Theory

Ship can work in different flow conditions for a towing tank test:

- Double model: only the wetted part of the ship is solved
- Fixed trim condition: two phase must be solved
- Free trim condition: moving boundary must be considered
- Self-propulsion: An equilibrium with the propeller must be considered

Due to the flow conditions the numerical techniques adopted must be changed:

- Double model: single phase steady solver
- Fixed trim condition: multi phase (LTSInterFoam / interFoam)
- Free trim condition: multi phase with dynamic mesh morphing (interDyNFoam)
- Self-propulsion: Sliding mesh… AMI!

Double model

The Double Model: can be solved with the simpleFoam solver

Very easy….

Fixed trim

Fixed trim: can be solved with the interFoam solver, but the solution is strongly depending to the local Courant Number…strongly depending to the mesh used.

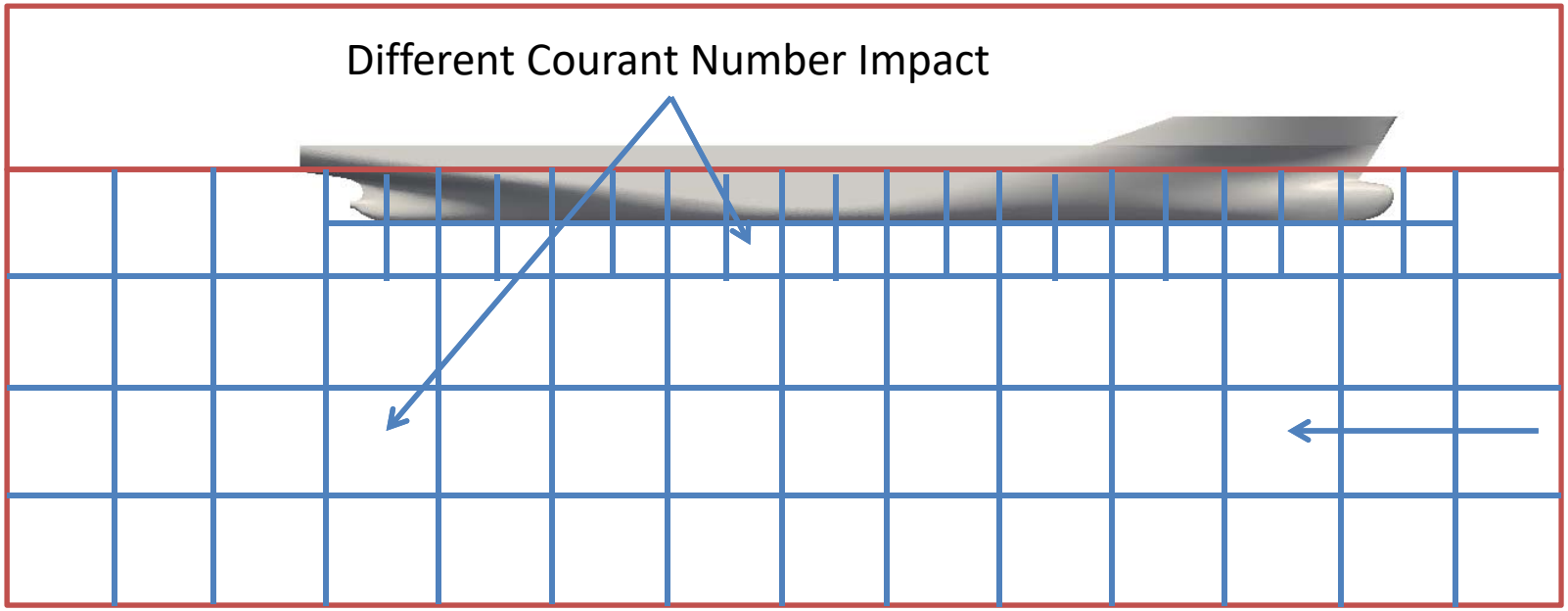
SnappyHexMesh commonly generate mesh with low Courant impact

Commercial code can generate worse mesh (from this point of view)

Fixed trim - LTS

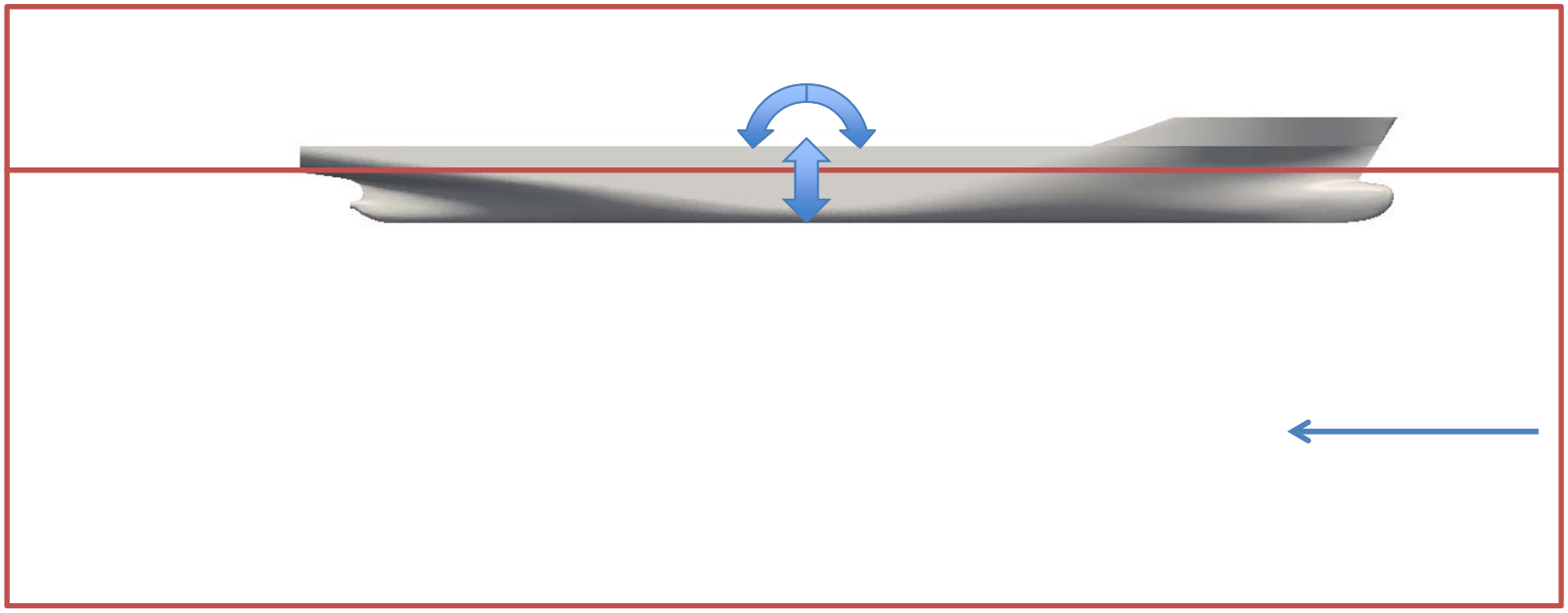
Fixed trim: can be also solved with the interFoam solver (time scheme: localEuler rDeltaT). This solver overcome the Courant limitation, but generate some more problem in terms of stability, moreover the solution is only steady.

LTS…Local Time Step: The main idea is to evolve the fields in time with the maximum time step for each cell.



Free trim

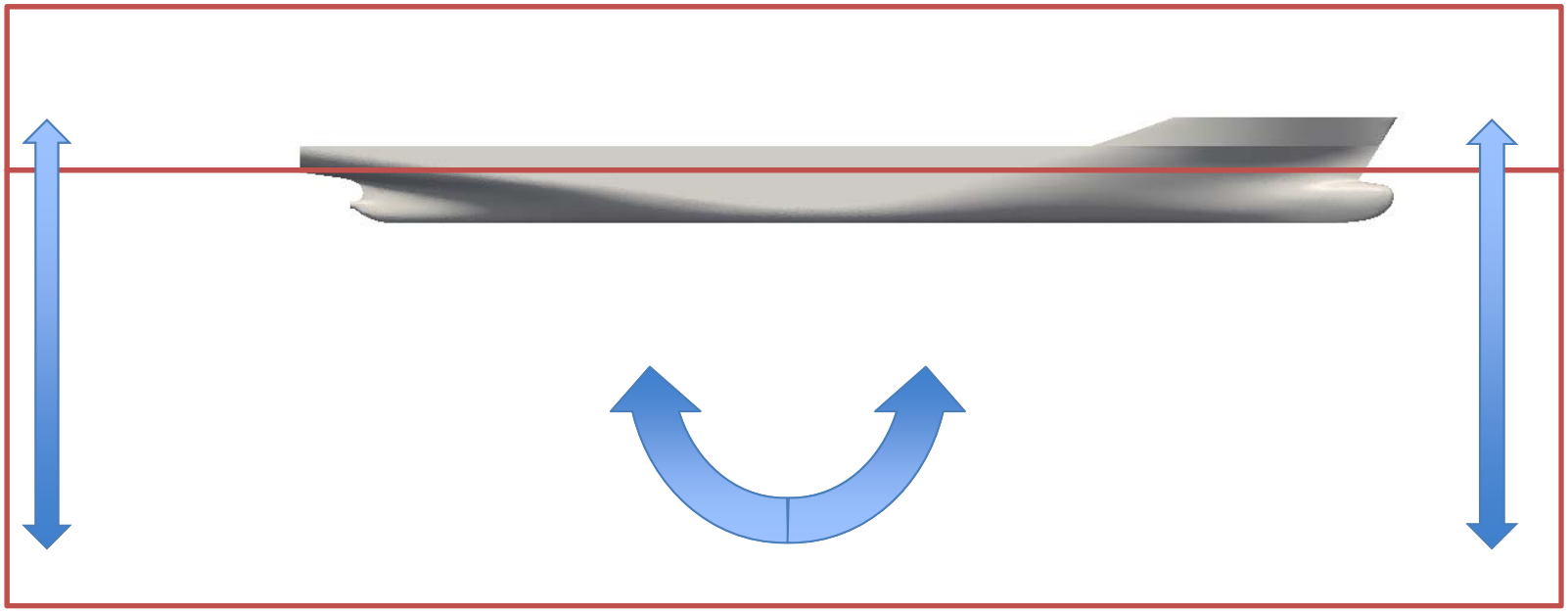
Free trim: can be solved with the InterDyMFoam solver. The ship must be free to move according to the equilibrium condition (Mass proprieties are important).



Free trim

Free trim: can be also solved with the InterDyMFoam solver. The ship must be free to move according to the equilibrium condition (Mass proprieties are important).
Different strategies can be adopted:

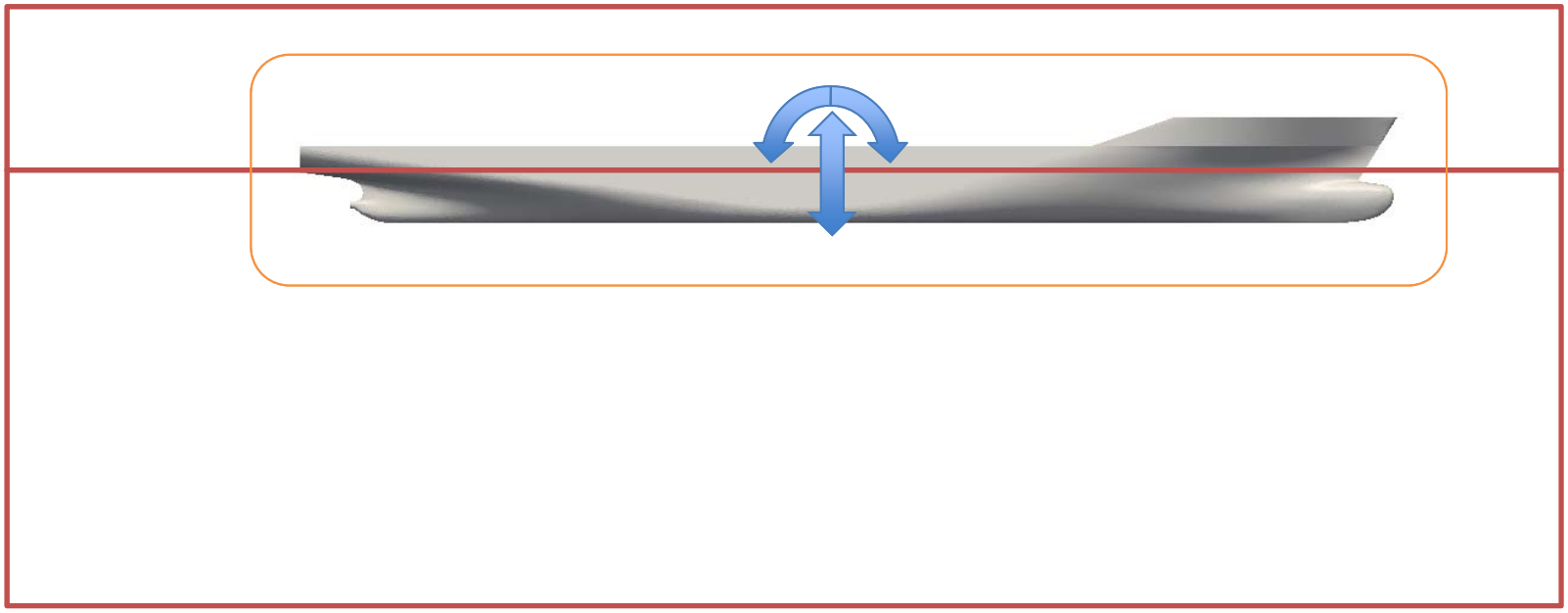
1. All the mesh rigidly moved: fast mesh motion, but more refinements must be setup.



Free trim

Free trim: can be also solved with the InterDyMFoam solver. The ship must be free to move according to the equilibrium condition (Mass proprieties are important).
Different strategies can be adopted:

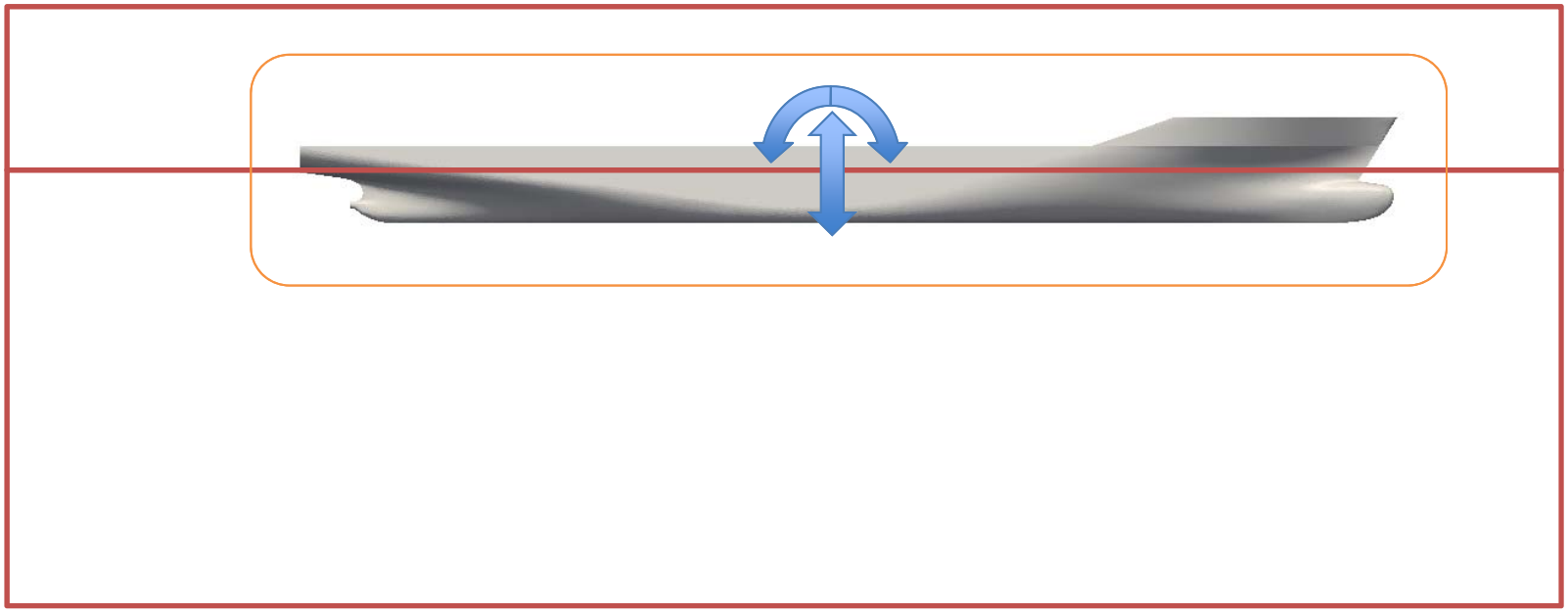
1. All the mesh rigidly moved
2. Morphing mesh: more computational effort



Free trim

Free trim: can be also solved with the InterDyMFoam solver. The ship must be free to move according to the equilibrium condition (Mass proprieties are important).
Different strategies can be adopted:

1. All the mesh rigidly moved
2. Morphing mesh
3. Chimera mesh (not still implemented in standard OF)



Today's lecture

1. **Introduction to the hydrodynamic naval problems**
    1. **Ship Resistance**
    2. **Propeller Performance**
    3. **Cavitation with OpenFOAM**
2. **Hands-on session.**
    1. **Drag resistance - KCS**
    2. **Open Water propeller performances - E779A**
    3. **Cavitation on a wing profile - NACA 0010**

Propeller Theory

Propeller can work in different flow conditions:

- Open water condition: propeller rotates in a undisturbed flow field and its rotation axes is parallel to the flow
- Inclined axes condition: propeller rotates in a undisturbed flow and its axes is inclined respect the main flow direction
- Behind a wake: propeller rotates in a disturbed flow field due to the hull or other appendage (pod, brackets, etc..)
- At zero speed: propeller rotates but the ship speed is zero

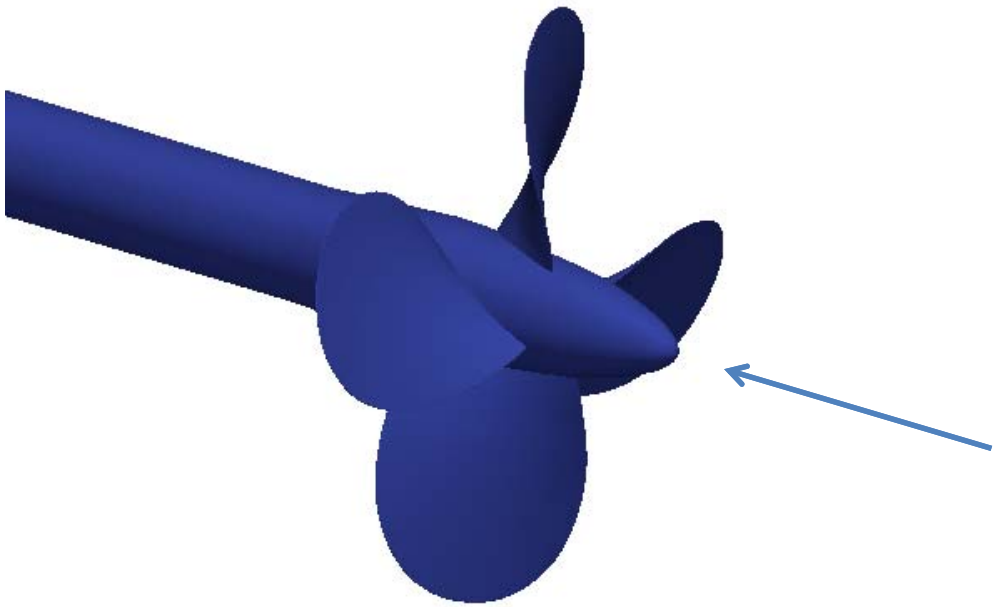
Due to the flow conditions the numerical techniques adopted must be changed:

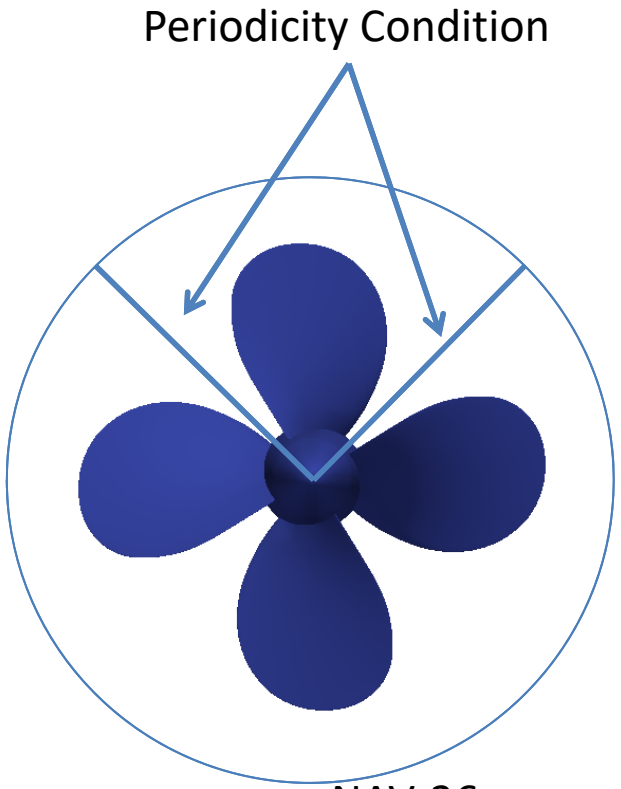
- Open water condition: the problem can be transformed in a steady problem
- Inclined axes condition: the problem is unsteady
- Behind a wake: unsteady problem and wake generation
- At zero speed: complicate boundary conditions

Open water condition

The problem can be simplified:

- Propeller has a periodic symmetry
- The OW condition respects the propeller symmetry
- In general the flow problem is unsteady (can it be make steady?)



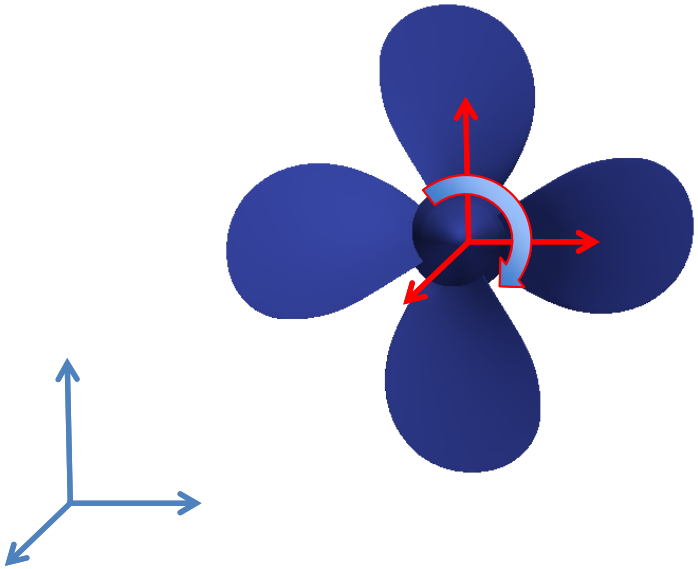


Open water condition

Why is the problem unsteady?

We can consider two reference systems :

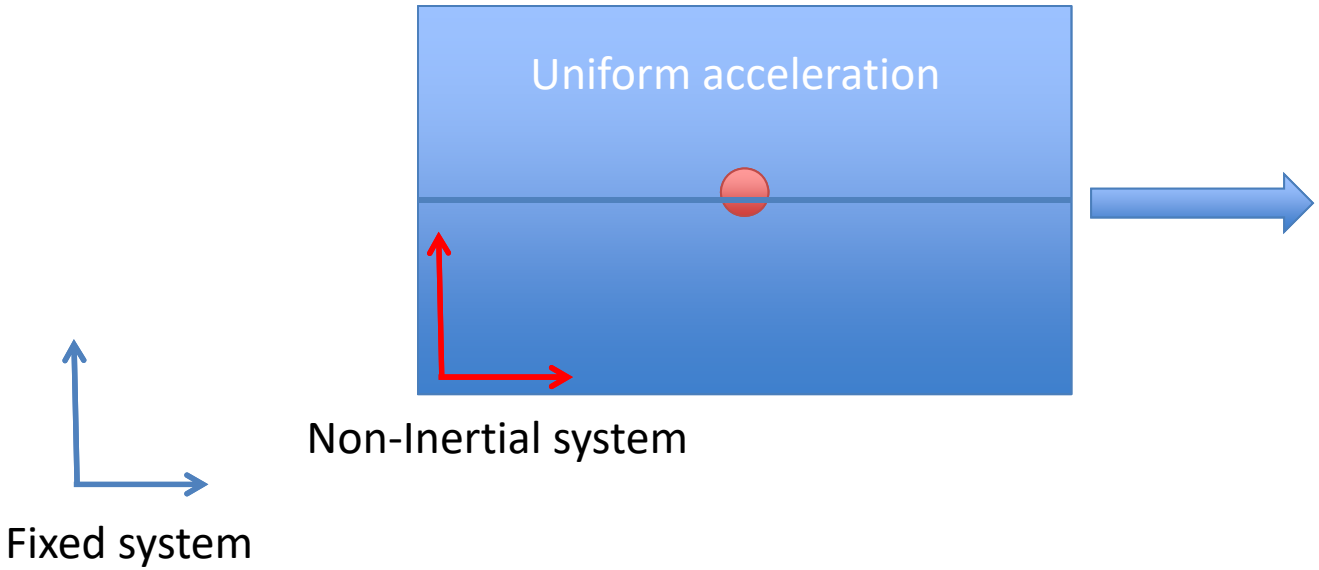
- First an inertial fixed system (fixed with the ship, world, stars…) → Classical flow equations
- Second a non-inertial system fixed with the blade → New flow equations MRF



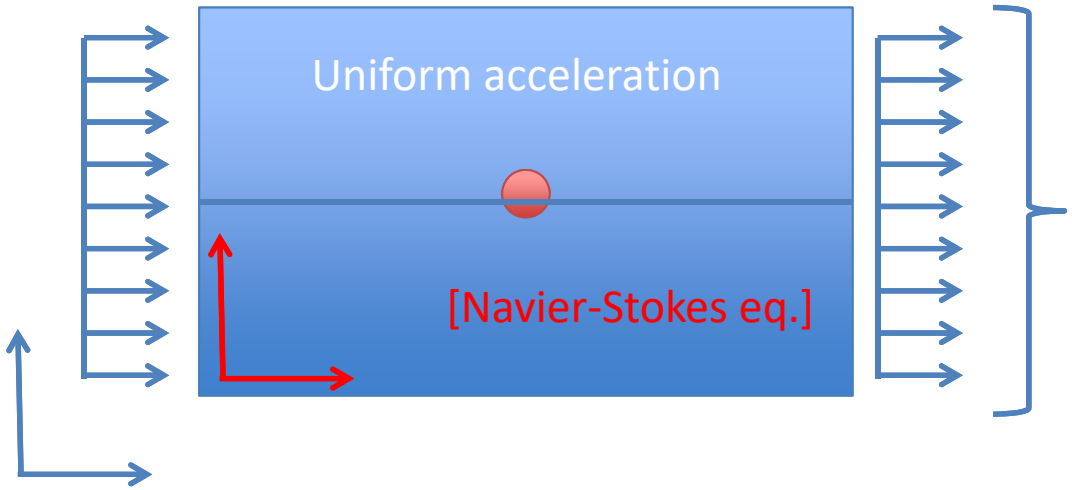
Open water condition – MRF approach

Why aren't the classical flow equations valid in a non-inertial reference frame?

In a problem with an uniform acceleration system what does a floating body experience?
How can I represent the problem?

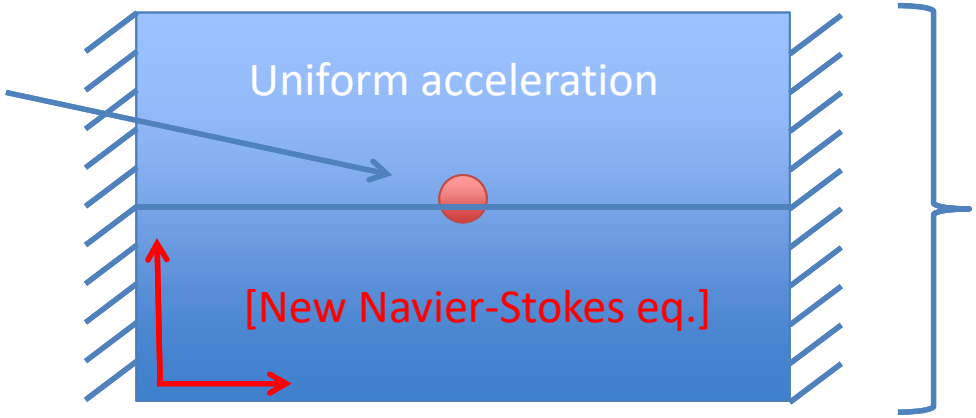


Open water condition – MRF approach



Not correct if the reference frame is fix with the box, but correct if R.F. is always fixed.

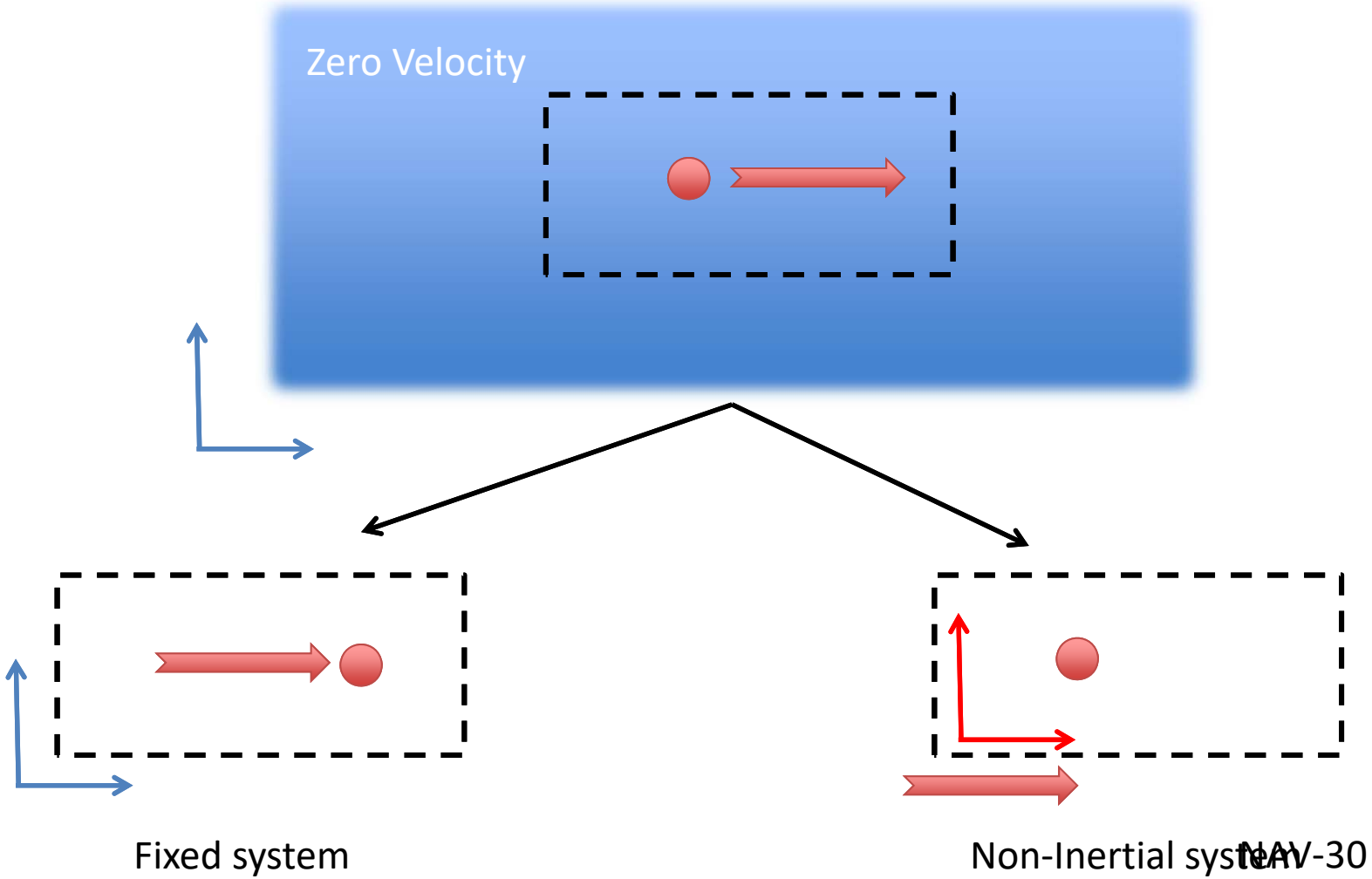
Due to the apparent force the sphere is subjected to a Mass force



Correct if the Navier-Stokes eq. are rewritten.

Open water condition – MRF approach

How can you setup a problem where an object accelerates in an infinite rest fluid?



Open water condition – MRF approach

Same problem for the propeller, therefore the equations can be rearranged in:

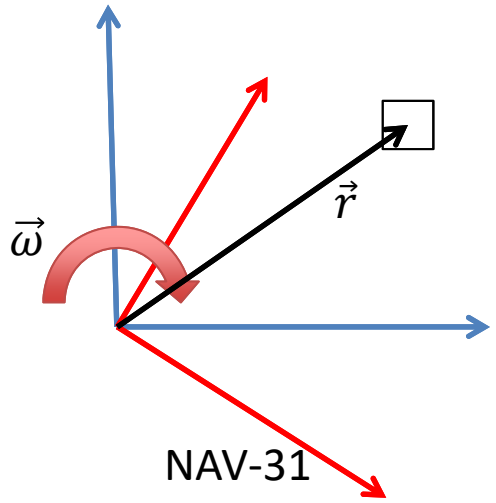
- Single Reference Frame (SRF)
- Multiple Reference Frame (MRF)

For a rotating reference frame centered in the axis origin we can write for the total derivative of a general vector:

$$\left.\frac{d\vec{A}}{dt}\right|_I = \left.\frac{d\vec{A}}{dt}\right|_R + \vec{\omega} \times \vec{A}$$

This equation applied for the position vector represents the relationship between the velocity vector in the two references

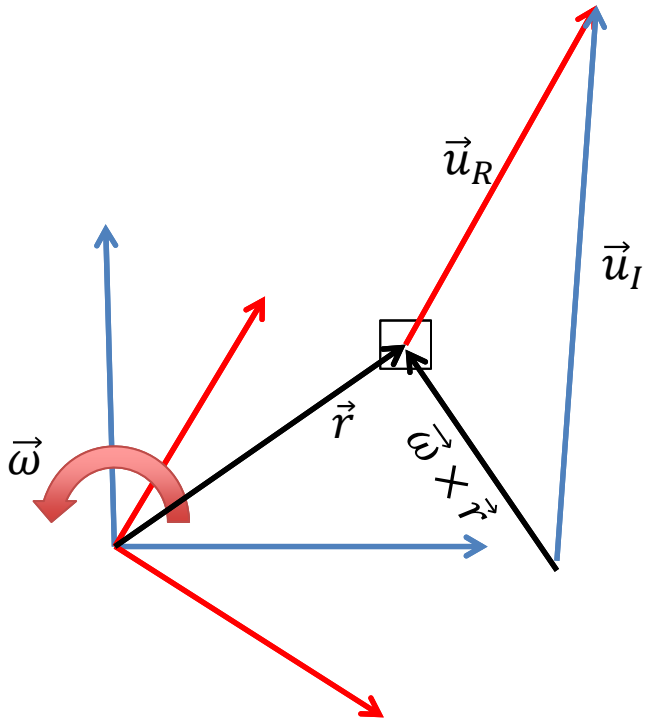
$$\vec{u}_I = \vec{u}_R + \vec{\omega} \times \vec{r}$$



Open water condition – MRF approach

Vectorial representation:

$$\vec{u}_I = \vec{u}_R + \vec{\omega} \times \vec{r}$$



Open water condition – MRF approach

If we extend for the acceleration vector the equation becomes :

$$\left.\frac{d\vec{u}_I}{dt}\right|_I = \left.\frac{d\vec{u}_I}{dt}\right|_R + \vec{\omega} \times \vec{u}_I$$

Replacing $\vec{u}_I = \vec{u}_R + \vec{\omega} \times \vec{r}$ we can write:

$$\left.\frac{d\vec{u}_I}{dt}\right|_I = \left.\frac{d[\vec{u}_R + \vec{\omega} \times \vec{r}]}{dt}\right|_R + \vec{\omega} \times [\vec{u}_R + \vec{\omega} \times \vec{r}]$$

$$\left.\frac{d\vec{u}_I}{dt}\right|_I = \left.\frac{d\vec{u}_R}{dt}\right|_R + \left.\frac{d\vec{\omega}}{dt} \times \vec{r} + \vec{\omega} \times \frac{d\vec{r}}{dt}\right|_R + \vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r}$$

$$\left.\frac{d\vec{u}_I}{dt}\right|_I = \left.\frac{d\vec{u}_R}{dt}\right|_R + \frac{d\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r}$$

Open water condition – MRF approach

The Navier-Stokes equations for an incompressible fluid in a inertial reference frame are:

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \underbrace{\dfrac{\partial \vec{u}_I}{\partial t} + \vec{u}_I \cdot \nabla \vec{u}_I}_{\dfrac{d\vec{u}_I}{dt}} = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

From previous:

$$\left.\frac{d\vec{u}_I}{dt}\right|_I = \underbrace{\left.\frac{d\vec{u}_R}{dt}\right|_R}_{\dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_R} + \frac{d\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r}$$

Open water condition – MRF approach

The Navier-Stokes equations for an incompressible fluid in a inertial reference frame are:

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \dfrac{\partial \vec{u}_I}{\partial t} + \vec{u}_I \cdot \nabla \vec{u}_I = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

Replacing $\vec{u}_I = \vec{u}_R + \vec{\omega} \times \vec{r}$ we can write:

$$\nabla \cdot \vec{u}_I = \nabla \cdot [\vec{u}_R + \vec{\omega} \times \vec{r}] = \nabla \cdot \vec{u}_R + \nabla \cdot [\vec{\omega} \times \vec{r}]$$

=0

$$\Delta \vec{u}_I = \Delta [\vec{u}_R + \vec{\omega} \times \vec{r}] = \Delta \vec{u}_R + \Delta [\vec{\omega} \times \vec{r}]$$

=0

Open water condition – MRF approach

The Navier-Stokes equations for an incompressible fluid in a inertial reference frame are:

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \dfrac{\partial \vec{u}_I}{\partial t} + \vec{u}_I \cdot \nabla \vec{u}_I = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

Replacing all the equations:

$$\begin{cases} \nabla \cdot \vec{u}_R = 0 \\ \dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_R + \underbrace{\dfrac{d\vec{\omega}}{dt} \times \vec{r}} + \underbrace{2\vec{\omega} \times \vec{u}_R} + \underbrace{\vec{\omega} \times \vec{\omega} \times \vec{r}} = -\nabla p_d + \nu \Delta \vec{u}_R \end{cases}$$

Centrifugal force

Coriolis force

=0 if the frame rotate with a constant velocity

Open water condition – MRF approach

Can I rewrite the equations with an absolute velocity?

$$\begin{cases} \nabla \cdot \vec{u}_R = 0 \\ \dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_R + \dfrac{d\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r} = -\nabla p_d + \nu \Delta \vec{u}_R \end{cases}$$

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_R + \dfrac{d\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r} = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

$$\vec{u}_R \cdot \nabla [\vec{u}_I - \vec{\omega} \times \vec{r}] = \vec{u}_R \cdot \nabla \vec{u}_I - \vec{u}_R \cdot \nabla [\vec{\omega} \times \vec{r}]$$

$$\vec{\omega} \times \vec{u}_R$$

Open water condition – MRF approach

Can I rewrite the equations with an absolute velocity?

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_I - \vec{\omega} \times \vec{u}_R + \dfrac{d\vec{\omega}}{dt} \times \vec{r} + 2\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r} = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

$$\vec{\omega} \times \vec{u}_R + \vec{\omega} \times \vec{\omega} \times \vec{r} = \vec{\omega} \times [\vec{u}_R + \vec{\omega} \times \vec{r}] = \vec{\omega} \times \vec{u}_I$$

Rewriting :

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \dfrac{\partial \vec{u}_R}{\partial t} + \vec{u}_R \cdot \nabla \vec{u}_I + \dfrac{d\vec{\omega}}{dt} \times \vec{r} + \vec{\omega} \times \vec{u}_I = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

Coriolis force

=0 if steady

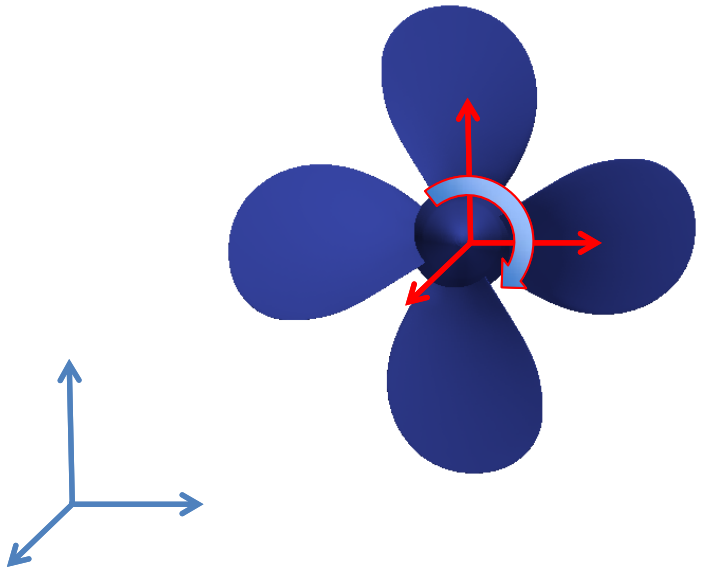
=0 if the frame rotate with a constant velocity

Open water condition – MRF approach

For a «steady» propeller the problem becomes:

$$\begin{cases} \nabla \cdot \vec{u}_I = 0 \\ \vec{u}_R \cdot \nabla \vec{u}_I + \vec{\omega} \times \vec{u}_I = -\nabla p_d + \nu \Delta \vec{u}_I \end{cases}$$

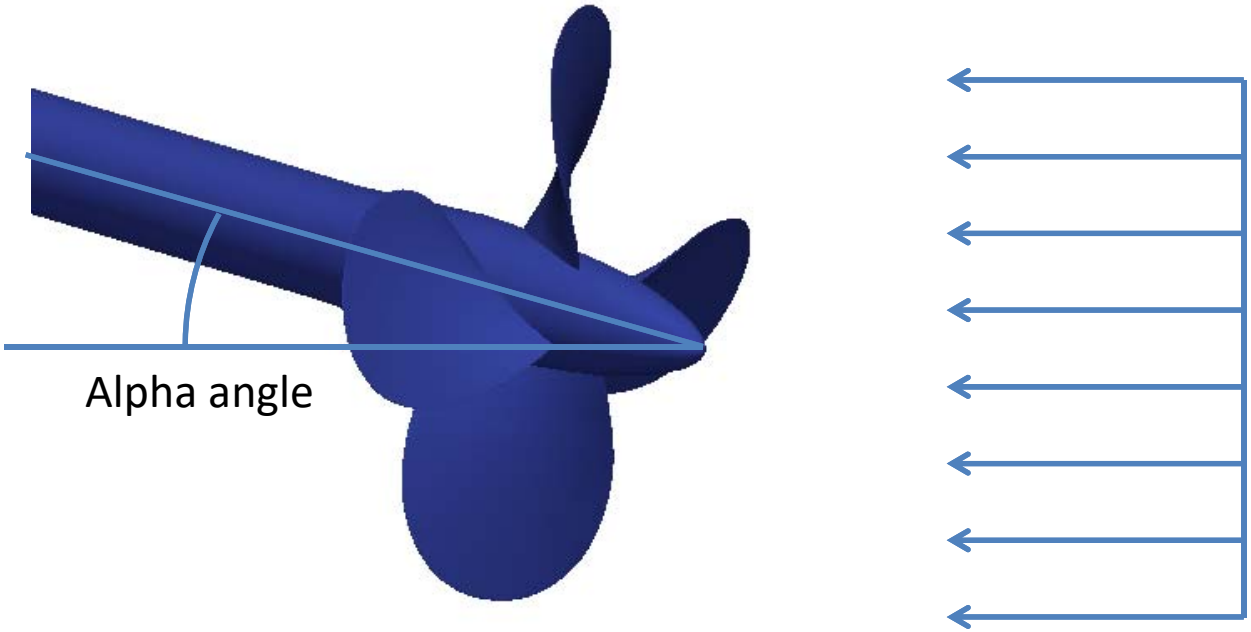
MRF approach can reduce an unsteady problem in a faster steady problem.



Inclined axes condition

Due to the non symmetry of the flow field no simplification can be adopted for the inclined axes propeller:

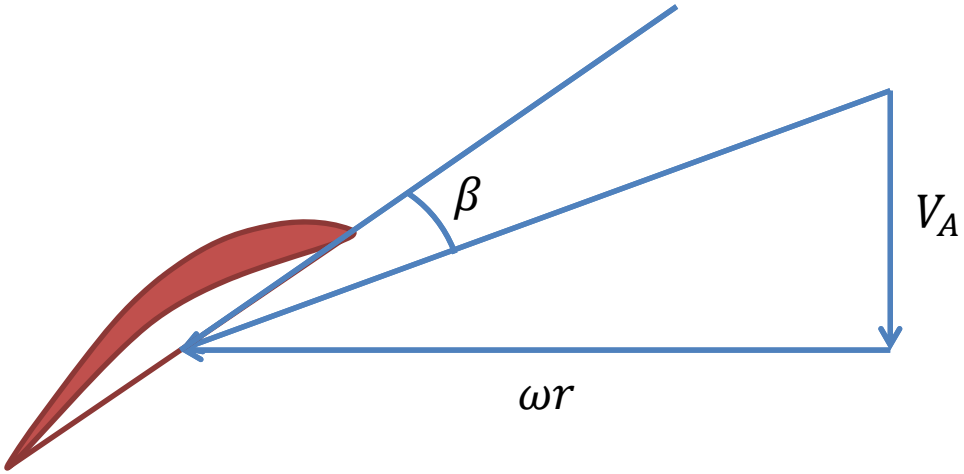
- The blade sees a different angle of attack during one revolution.

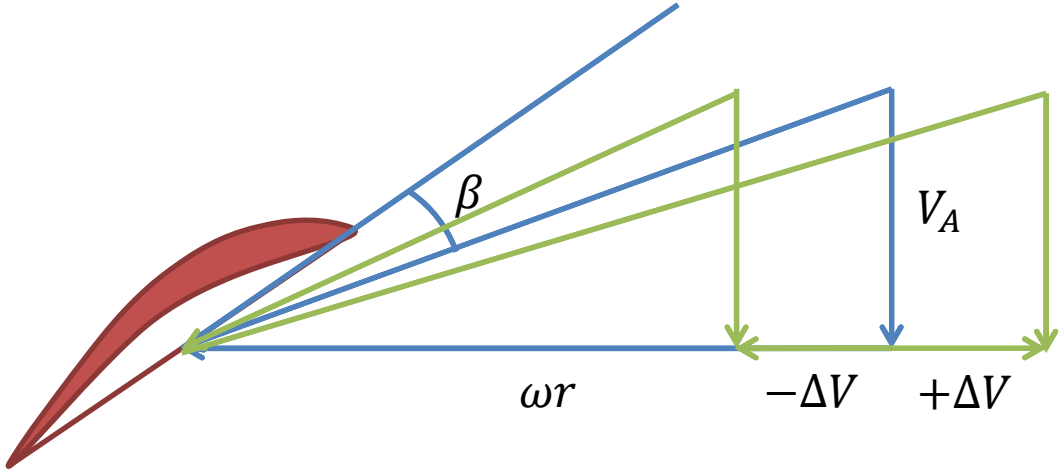


Inclined axes condition

Due to the non symmetry of the flow field no simplification can be adopted for the inclined axes propeller:

- The blade sees a different angle of attack during one revolution.



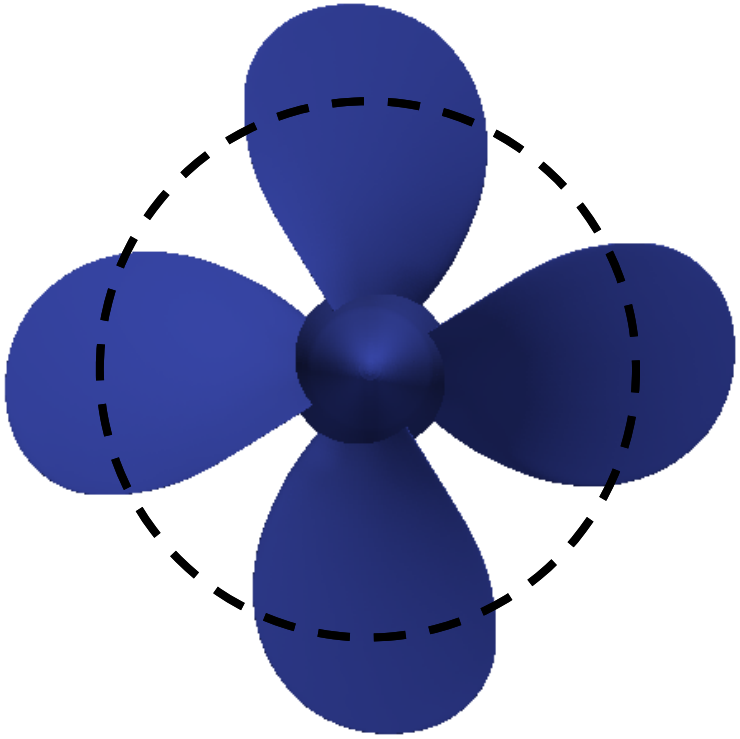


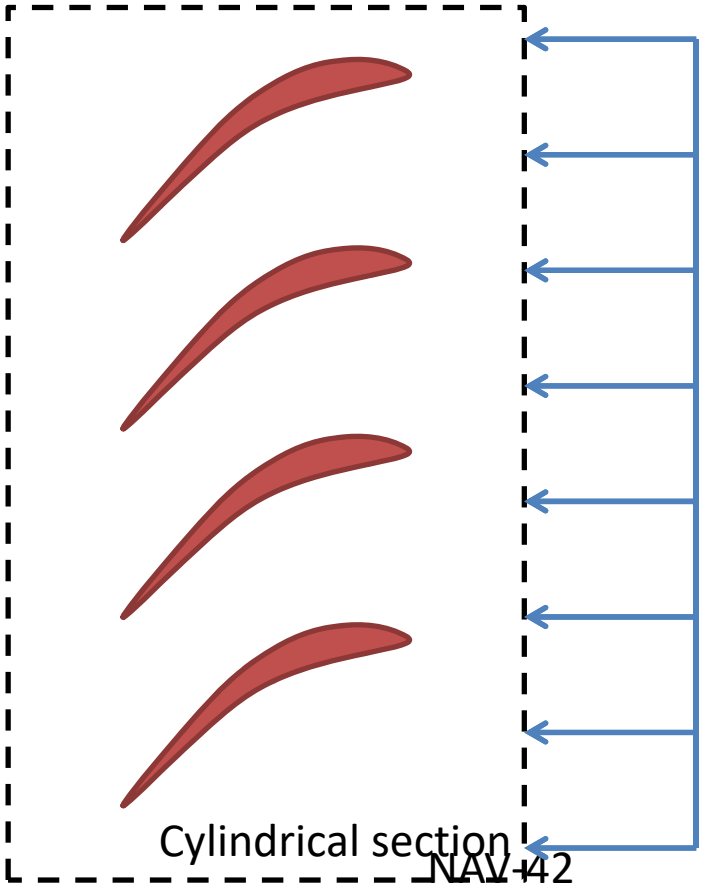
Induced velocity due to the inclined axes

Inclined axes condition

Due to the non symmetry of the flow field no simplification can be adopted for the inclined axes propeller:

- The blade sees a different angle of attack during one revolution.
- The performance of each blade is strongly influenced by the other blades, therefore all the blades must be solved simultaneously



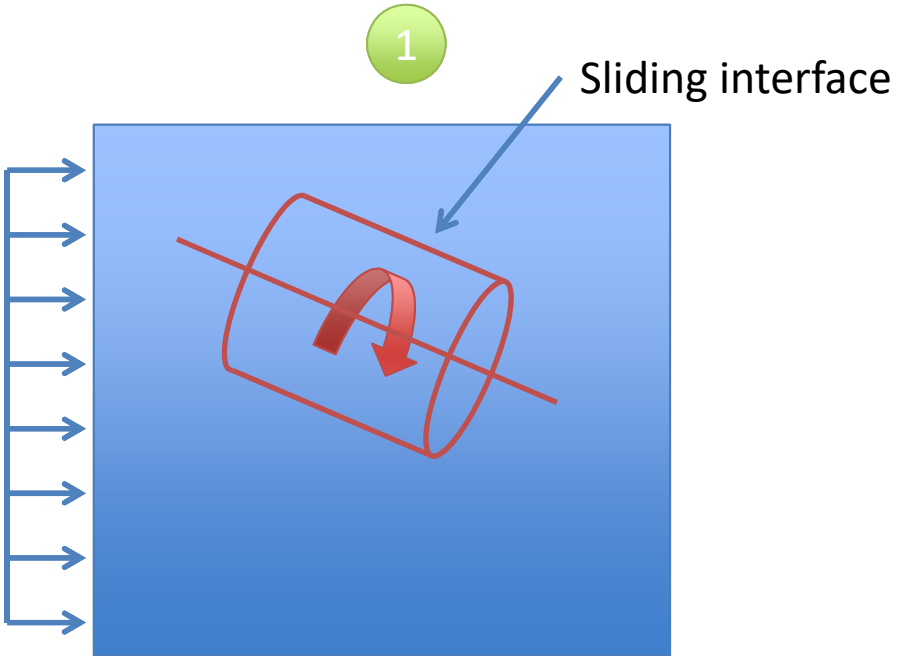


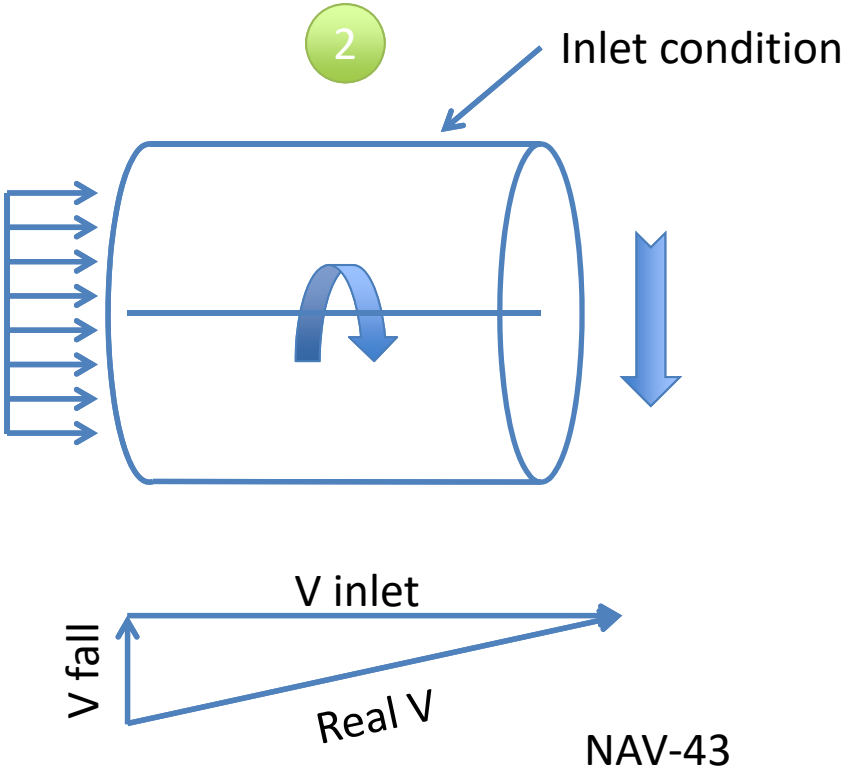
Inclined axes condition

For propeller in this working condition an unsteady solver is necessary, therefore an increase of the computational time is needed.

Two strategies can be adopted:

1. Inclined cylindrical domain sliding in a undisturbed domain
2. Falling cylindrical domain

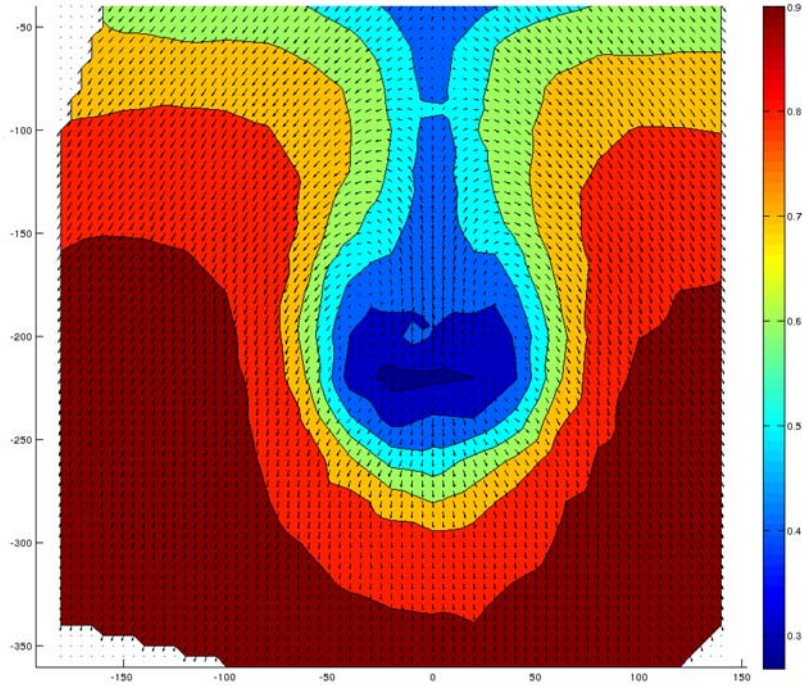




Behind a wake condition

No domain or solver strategies can be adopted to speed up the solution:

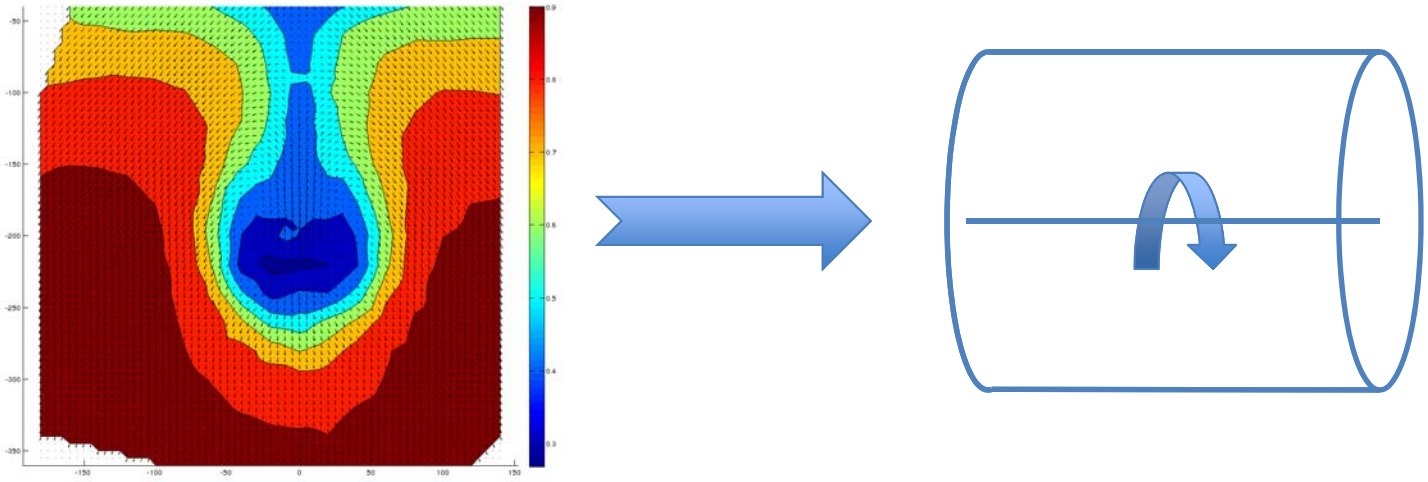
- Full unsteady and all the blades must be solved simultaneously
- New problem: How can we create the needed hull wake?



Behind a wake condition

No domain or solver strategies can be adopted to speed up the solution:

- Full unsteady and all the blades must be solved simultaneously
- New problem: How can we create the needed hull wake?
  1. Imposing the wake at the inlet boundary:
     - ⇧ Very simple to implement (not so much in OF)
     - ⇩ The imposed wake is not hydro-dynamically stable, the wake can disappear

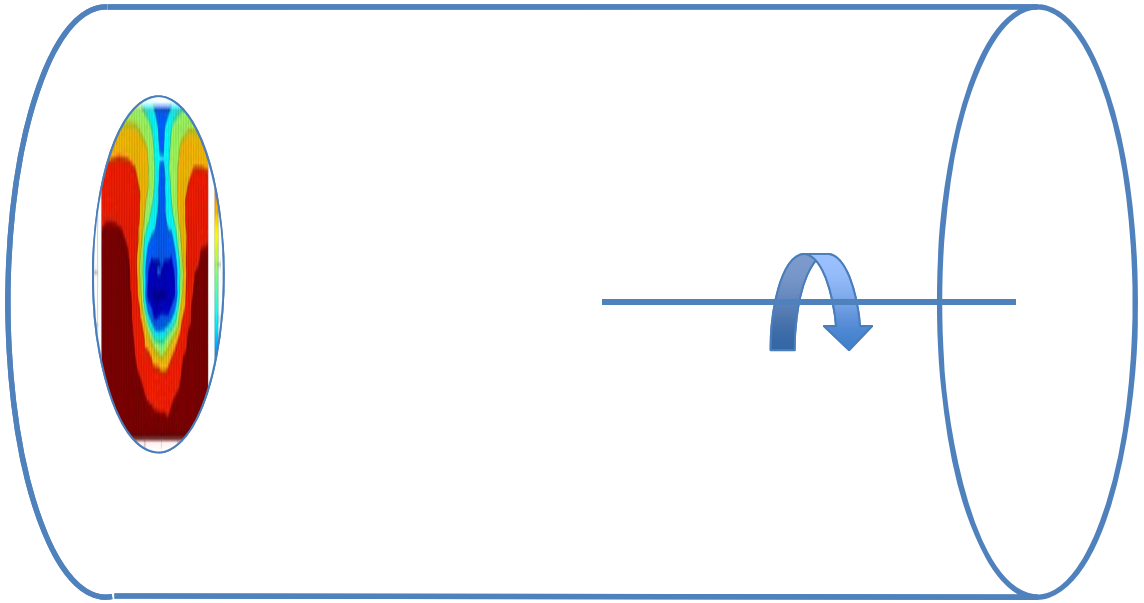


Behind a wake condition

No domain or solver strategies can be adopted to speed up the solution:

- Full unsteady and all the blades must be solved simultaneously
- New problem: How can we create the needed hull wake?
  1. Imposing the wake at the inlet boundary
  2. Using porous baffle or body force to generate the correct wake
     - ⇧ The generated wake is stable and real
     - ⇩ How to set the correct porous baffle or force distribution configuration is a problem

Same effect as the wake screen used in a cavitation tunnel.



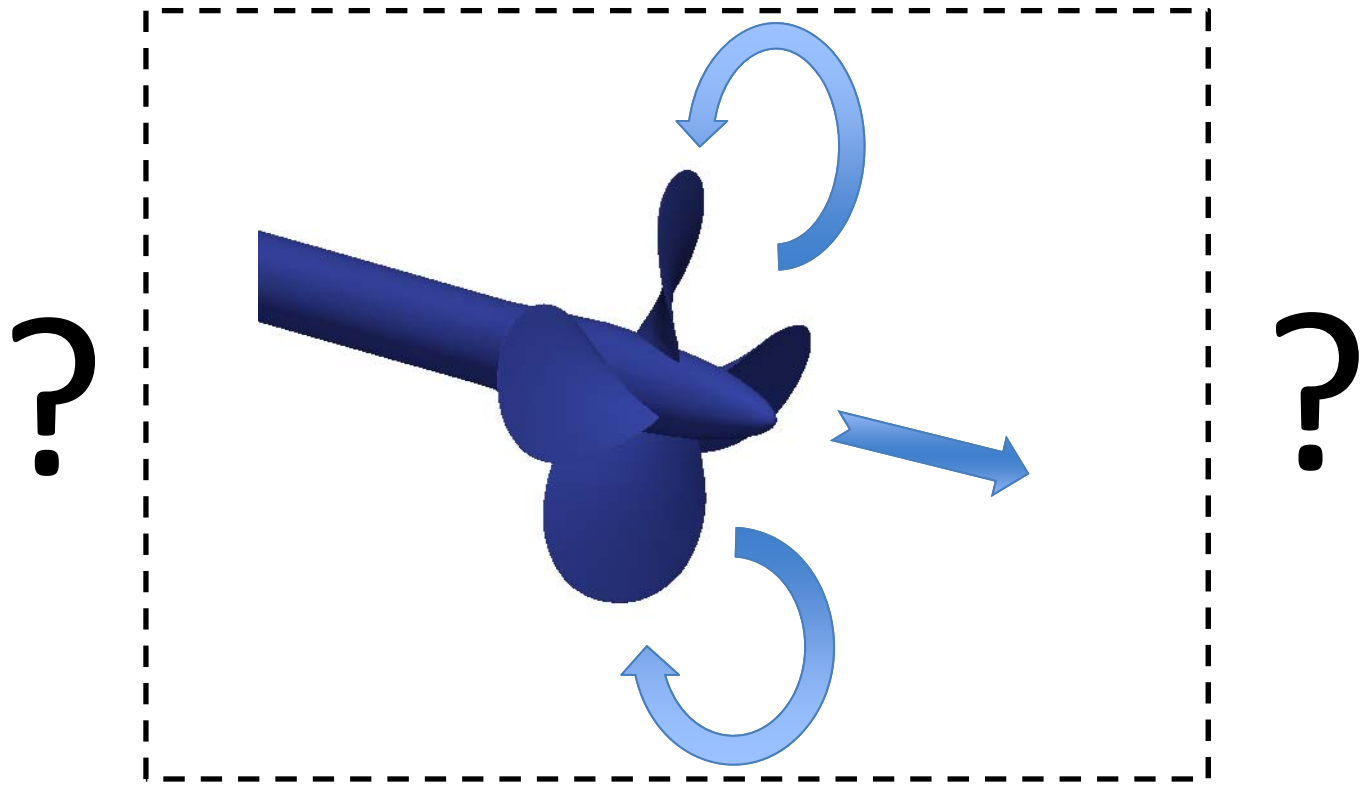
Behind a wake condition

No domain or solver strategies can be adopted to speed up the solution:

- Full unsteady and all the blades must be solved simultaneously
- New problem: How can we create the needed hull wake?
  1. Imposing the wake at the inlet boundary
  2. Using porous baffle or body force to generate the correct wake
  3. Discretize the entire hull or a portion of it

Zero Speed

At Zero speed condition the propeller generates complicated flow fields and it is in a very off design condition, therefore the correct boundary conditions are strongly dependent on the chosen configuration

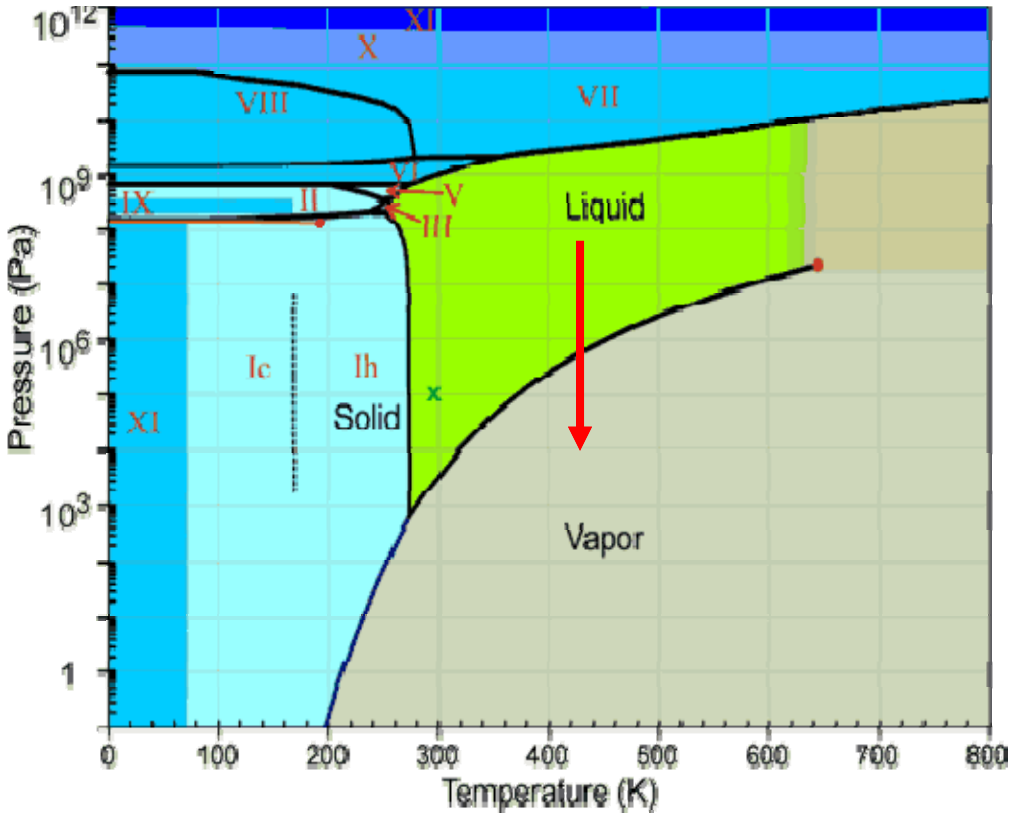


Today's lecture

1. **Introduction to the hydrodynamic naval problems**
    1. **Ship Resistance**
    2. **Propeller Performance**
    3. **Cavitation with OpenFOAM**
2. **Hands-on session.**
    1. **Drag resistance - KCS**
    2. **Open Water propeller performances - E779A**
    3. **Cavitation on a wing profile - NACA 0010**

Cavitation

- Cavitation is the passage from liquid to vapour achieved by reducing pressure instead than by increasing temperature



http://www.phy.duke.edu/~hsg/363/table-images/water-phase-diagram.html

Cavitation

- A well-established approach is based on the *homogeneous mixture* idealization of a multiphase flow

- The cavitating flow is a mixture of two species, vapour and liquid, behaving as one.

- Phases are considered incompressible and share the same instantaneous velocity **U** and pressure fields p, solved with the *VoF* approach

Cavitation

- The homogeneous mixture approach is exactly the same adopted for free surface problems,
- An appropriate source term is needed to account for the phase passage, depending on the variations of pressure,
- The modeling of this interphase mass transfer rate is the core of any cavitation model

Cavitation

- RANS Equation for the homogeneous mixture can be derived as (Zwart, 2004):

$$\begin{cases} \nabla \cdot \boldsymbol{U} = \dot{m}\left(\dfrac{1}{\rho_l} - \dfrac{1}{\rho_v}\right) \\ \dfrac{\partial(\rho \boldsymbol{U})}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = -\nabla P - \nabla \cdot \boldsymbol{\tau} + S_M \\ \dfrac{\partial \gamma}{\partial t} + \nabla \cdot (\gamma \boldsymbol{U}) = \dfrac{\dot{m}}{\rho_l} \end{cases}$$

The continuity equation is corrected with a source term, where $\dot{m}$ is the interphase mass transfer rate

Zwart, P., Gerber, A. G. & Belamri, T. (2004).'A Two-Phase Model for Predicting Cavitation Dynamics'. ICMF 2004 International Conference on Multiphase Flow, Yokohama, Japan.

Cavitation

- RANS Equation for the homogeneous mixture can be derived as (Zwart, 2004):

$$\begin{cases} \nabla \cdot \boldsymbol{U} = \dot{m}\left(\dfrac{1}{\rho_l} - \dfrac{1}{\rho_v}\right) \\ \dfrac{\partial(\rho \boldsymbol{U})}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = -\nabla P - \nabla \cdot \boldsymbol{\tau} + S_M \\ \dfrac{\partial \gamma}{\partial t} + \nabla \cdot (\gamma \boldsymbol{U}) = \dfrac{\dot{m}}{\rho_l} \end{cases}$$

The continuity equation is corrected with a source term, where $\dot{m}$ is the interphase mass transfer rate

Also the transport equation for the fraction of the liquid phase is corrected

Zwart, P., Gerber, A. G. & Belamri, T. (2004).'A Two-Phase Model for Predicting Cavitation Dynamics'. ICMF 2004 International Conference on Multiphase Flow, Yokohama, Japan.

Cavitation

- The Volume of Fluid approach is used to solve the set of equations:

$$\gamma = \frac{volume\ liquid}{total\ volume} \qquad \alpha = \frac{volume\ vapour}{total\ volume}$$

$$\gamma + \alpha = 1$$

$$\begin{cases} \rho = \gamma\rho_l + (1-\gamma)\rho_v \\ \mu = \gamma\mu_l + (1-\gamma)\mu_v \end{cases}$$

Cavitation

- The modeling of the interphase mass transfer rate is the core of any cavitation model.

- Within the OpenFOAM environment, the class of solver for incompressible cavitating flows is the:

`interPhaseChange(DyM)Foam`

- The **`DyM`** is necessary when dynamic meshes are accounted for.

Cavitation

- OpenFOAM has three models for $\dot{m}$:
  - **Merkle**
  - **Kunz**
  - **Schnerr-Sauer**

That can be selected in the `transportProprieties` file under the `/constant` folder.

Cavitation

- OpenFOAM has three models for $\dot{m}$, which parameters can be tuned :

```
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 phases (water vapour);
18
19 phaseChangeTwoPhaseMixture SchnerrSauer;
20
21 pSat             [1 -1 -2 0 0 0 0] 3200;    // Saturation pressure
22
23 sigma            [1 0 -2 0 0 0 0]  0.07;
24
25 water
26 {
27     transportModel Newtonian;
28     nu              [0 2 -1 0 0 0 0] 1e-06;
29     rho             [1 -3 0 0 0 0 0] 1000;
30 }
31
32 vapour
33 {
34     transportModel Newtonian;
35     nu              [0 2 -1 0 0 0 0] 4.273e-04;
36     rho             [1 -3 0 0 0 0 0] 0.02308;
37 }
38
39 KunzCoeffs
```

Names of the phases

Selected cavitation model

Surface tension is significant when *Ca <<1* or *We <<1*

$Re << 1$ $\qquad$ $Re >> 1$

$$Ca = \frac{\mu \mathbf{U}}{\sigma} \qquad We = \frac{\rho L \mathbf{U}^2}{\sigma}$$

Physical proprieties of the phases

Cavitation

- OpenFOAM has three models for $\dot{m}$, which parameters can be tuned :

```
KunzCoeffs
{
    UInf            UInf    [0 1 -1 0 0 0 0]    20.0;
    tInf            tInf    [0 0 1 0 0 0 0]     0.005; // L = 0.1 m
    Cc              Cc      [0 0 0 0 0 0 0]     1000;
    Cv              Cv      [0 0 0 0 0 0 0]     1000;
}

MerkleCoeffs
{
    UInf            UInf    [0 1 -1 0 0 0 0]    20.0;
    tInf            tInf    [0 0 1 0 0 0 0]     0.005;  // L = 0.1 m
    Cc              Cc      [0 0 0 0 0 0 0]     80;
    Cv              Cv      [0 0 0 0 0 0 0]     1e-03;
}

SchnerrSauerCoeffs
{
    n               n       [0 -3 0 0 0 0 0]    1.6e+13;
    dNuc            dNuc    [0 1 0 0 0 0 0]     2.0e-06;
    Cc              Cc      [0 0 0 0 0 0 0]     1;
    Cv              Cv      [0 0 0 0 0 0 0]     1;
}
```

Coefficients of the Mass transfer models

Cavitation

- **Merkle/Kunz** models are very similar. The Kunz model is an evolution of the original Merkle approach:

$$\dot{m} = \dot{m}^{+} + \dot{m}^{-}$$

$$\begin{cases} \dot{m}^{-} = \dfrac{C_{m\_\mathrm{dest}}\rho_l\rho_l \mathrm{MIN}(p - p_v, 0)(1 - \alpha_v)}{(0.50\rho_l U_\infty^2)\rho_v \; t_\infty}, & p < p_v, \\ \dot{m}^{+} = \dfrac{C_{m\_\mathrm{prod}}\rho_l \mathrm{MAX}(p - p_v, 0)\alpha_v}{(0.50\rho_l U_\infty^2)\, t_\infty}, & p > p_v. \end{cases}$$

Cavitation

- **Merkle/Kunz** models are very similar. The Kunz model is an evolution of the original Merkle approach:

$$\dot{m} = \dot{m}^{+} + \dot{m}^{-}$$

$$\begin{cases} \dot{m}^{-} = \dfrac{C_{m\_dest}\rho_l\rho_l \mathrm{MIN}(p - p_v, 0)(1 - \alpha_v)}{(0.50\rho_l U_\infty^2)\rho_v \; t_\infty}, & p < p_v, \\ \dot{m}^{+} = \dfrac{C_{m\_prod}\rho_l \mathrm{MAX}(p - p_v, 0)\alpha_v}{(0.50\rho_l U_\infty^2)\, t_\infty}, & p > p_v. \end{cases}$$

- Tuning Parameters: Cm_dest and Cm_prod, respectively for evaporation and condensation (`Cv` and `Cc` in `transportProprieties`)

Cavitation

- **Merkle/Kunz** models are very similar. The Kunz model is an evolution of the original Merkle approach:

- Suggested value:

  Cc = 80

  Cv = 0.001

- U_∞ and t_∞ are the reference speed and the characteristical time scale of the problem under investigation

Cavitation

- **Kunz** model:

$$\dot{m} = \dot{m}^{+} + \dot{m}^{-} \qquad \begin{cases} \dot{m}^{+} = \dfrac{C_{prod}\rho_v \gamma^2 (1-\gamma)}{t_\infty} \\ \dot{m}^{-} = \dfrac{C_{dest}\rho_v \gamma \min[0, P - P_v]}{(0.5\rho_l U_\infty^2) t_\infty} \end{cases}$$

- Suggested value:

  Cc = 200 - 500

  Cv = 4000 - 10000

- U_∞ and t_∞ as in Merkle

Cavitation

- For both models, the vaporization (production of the vapour phase) is proportional to the amount by which the pressure is below the vapour pressure.

- Condensation and vaporization are not symmetrical

Cavitation

- **Schnerr-Sauer** model:

$$\dot{m} = 3\frac{\rho_{vap}\rho_{liq}}{\rho_m}\frac{\alpha(1-\alpha)}{R}\sqrt{\frac{2}{3}\frac{|p-p_{vap}|}{\rho_{liq}}}\mathrm{sgn}\left(p_{vap}-p\right)$$

- Symmetrical condensation/vaporization
- Tuning parameters for the definition of the vapour fraction α (and the correspondent transport equation)

Cavitation

- **Schnerr-Sauer** model:

$$\frac{\partial \gamma}{\partial t} + \nabla \cdot (\gamma \boldsymbol{U}) = \frac{\dot{m}}{\rho_l} \qquad \text{or} \qquad \frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{U}) = \frac{\dot{m}}{\rho_v}$$

where:

$$\alpha = \frac{n_0 (4/3) \pi R^3}{1 + n_0 (4/3) \pi R^3}$$

Cavitation

- **Schnerr-Sauer** model:
- n_0 is the vapour nuclei concentration per unit volume (N/m^3),
- R is the initial radius of the nuclei.
- Suggest values:

  n_0 = 10^9 – 10^{13}

  R = 10^{-4} – 10^{-8}

Cavitation

- **Schnerr-Sauer** model:

- In OpenFOAM, custom uneven Schnerr-Sauer model with Cc, Cv parameters to unbalance vaporization with respect to condensation

```
SchnerrSauerCoeffs
{
    n                   n       [0 -3 0 0 0 0 0]     1.6e+13;
    dNuc                dNuc    [0 1 0 0 0 0 0]      2.0e-06;
    Cc                  Cc      [0 0 0 0 0 0 0]      1;
    Cv                  Cv      [0 0 0 0 0 0 0]      1;
}
```

Today's lecture

1. **Introduction to the hydrodynamic naval problems**
    1. **Ship Resistance**
    2. **Propeller Performance**
    3. **Cavitation with OpenFOAM**
2. **Hands-on session.**
    1. **Drag resistance - KCS**
    2. **Open Water propeller performances - E779A**
    3. **Cavitation on a wing profile - NACA 0010**

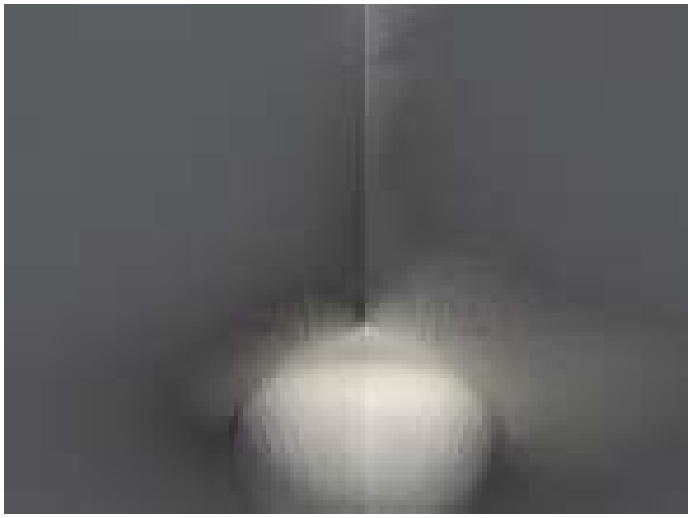
Hands-on session

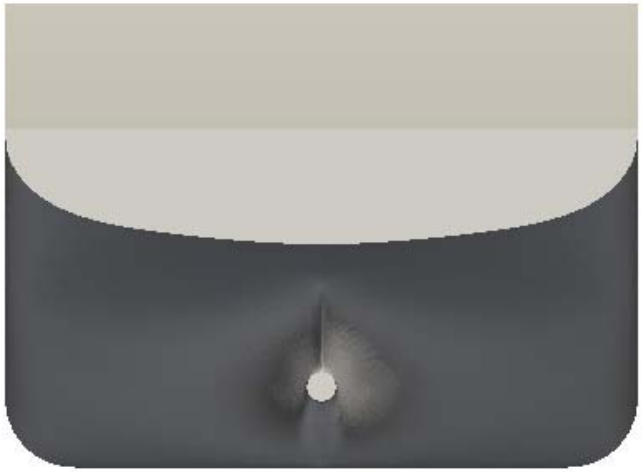
Today's lecture

1. Introduction to the hydrodynamic naval problems
    1. Ship Resistance
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

Hands-on session

KCS test case

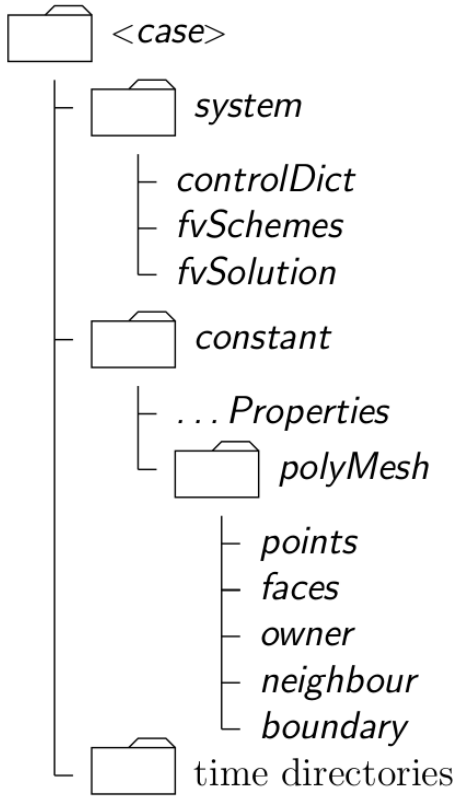




Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files: as seen in the previous lessons, for simplicity the initial case can be upload from the ….

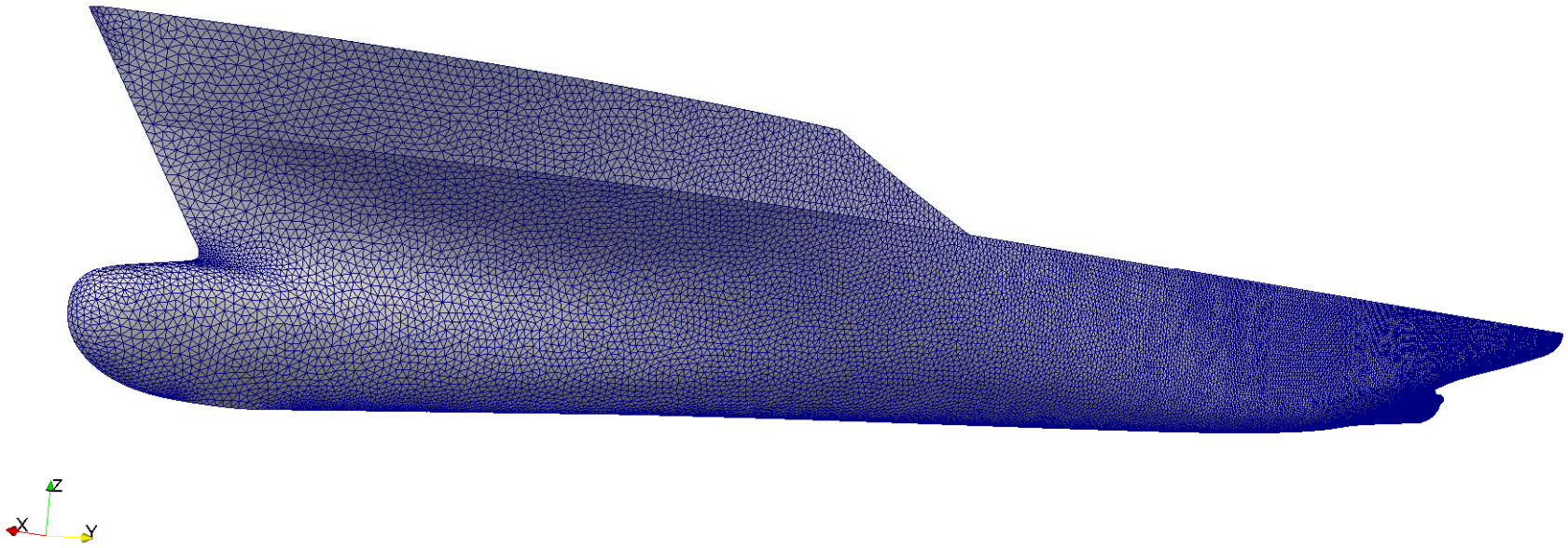


Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file: For our hands-on session the file can be found in KCS/constant/triSurface/Carena.stl.

The file should accurately represent the hull, if poor quality mesh has been used the solution may be not optimal.

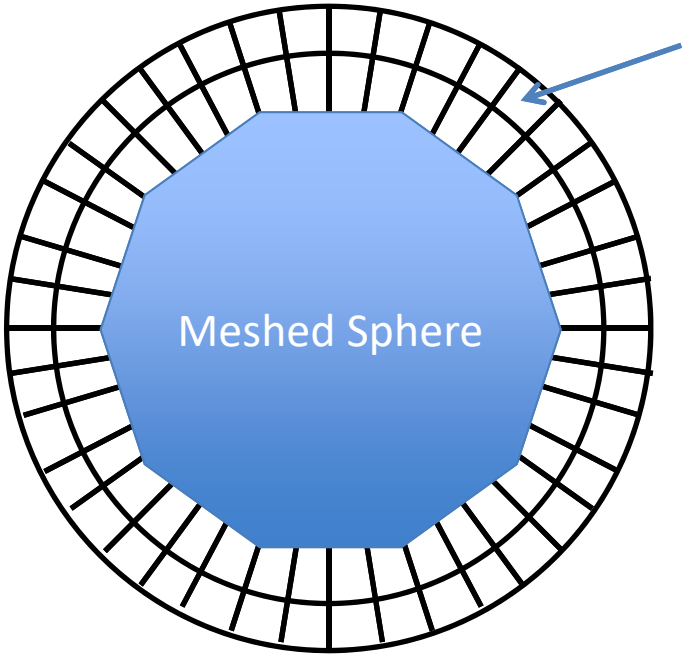


Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file: For our hands-on session the file can be found in KCS/constant/triSurface/Carena.stl.

The file should accurately represent the hull, if poor quality mesh has been used the solution may be not optimal.



The volume cells are smaller than the surface cells, therefore the steps are solved as real corner.

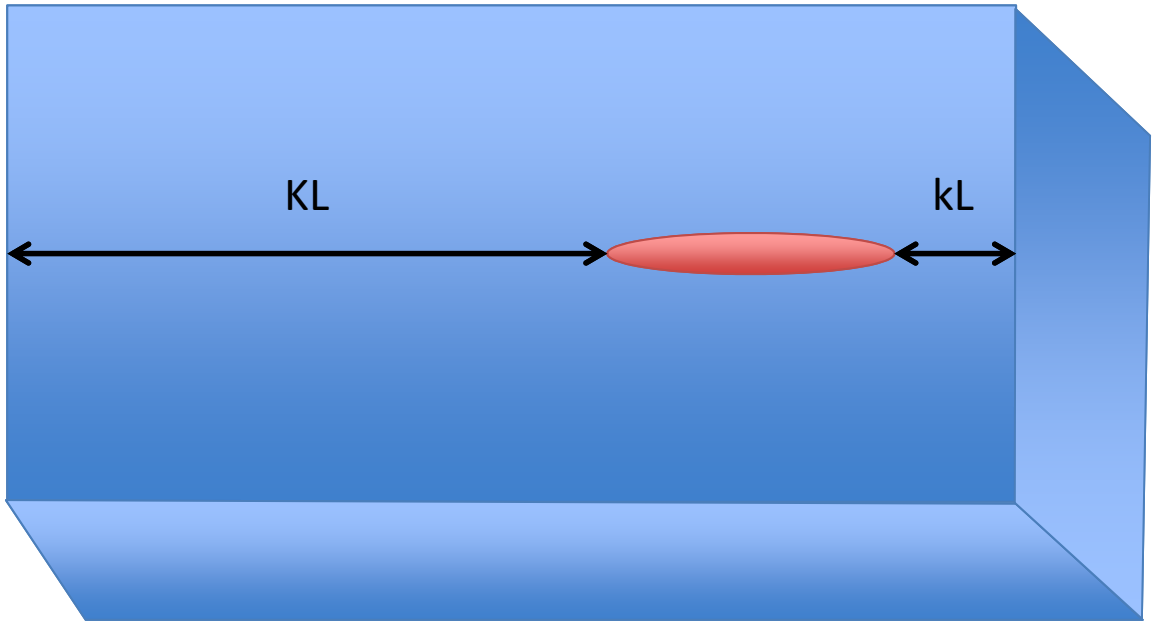
Consequences:

- pressure jumps
- Increasing of the shear stress tensor
- Increasing of turbulences

Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions: Some empirical rules can be followed to correctly define a domain
   - The domain shouldn't influence the hull drag (different if shallow water condition or confined water conditions are taken in to account)

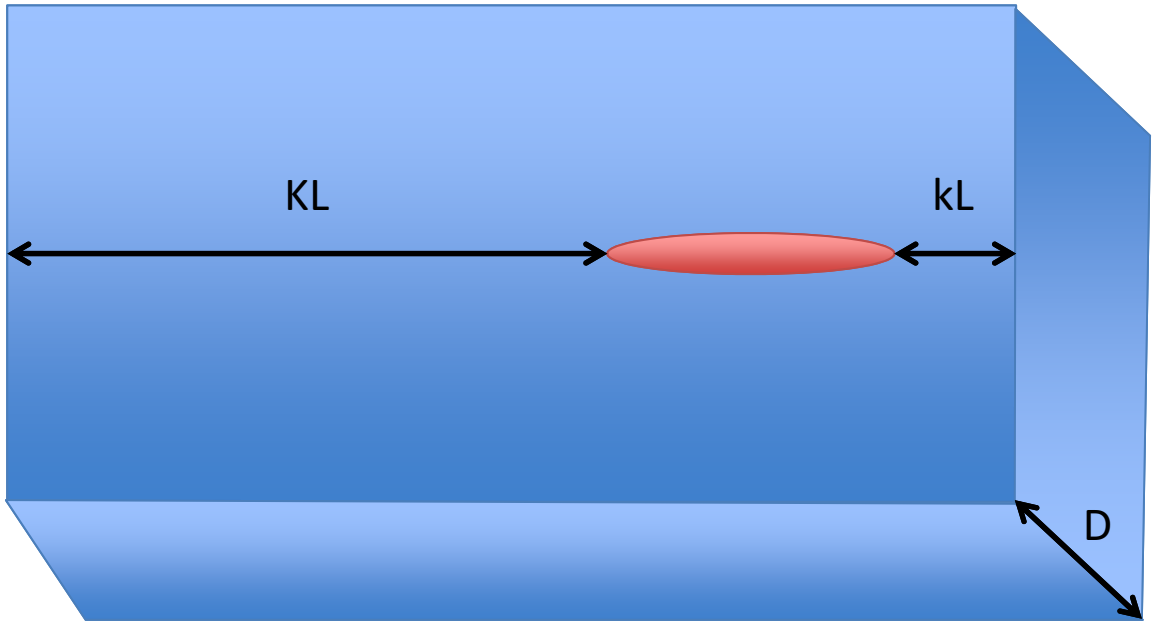


Commonly k can be set equal to 1.5-2 and K>>k!

Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions: Some empirical rules can be followed to correctly define a domain
   - The domain shouldn't influence the hull drag (different if shallow water condition or confined water conditions are taken in to account)



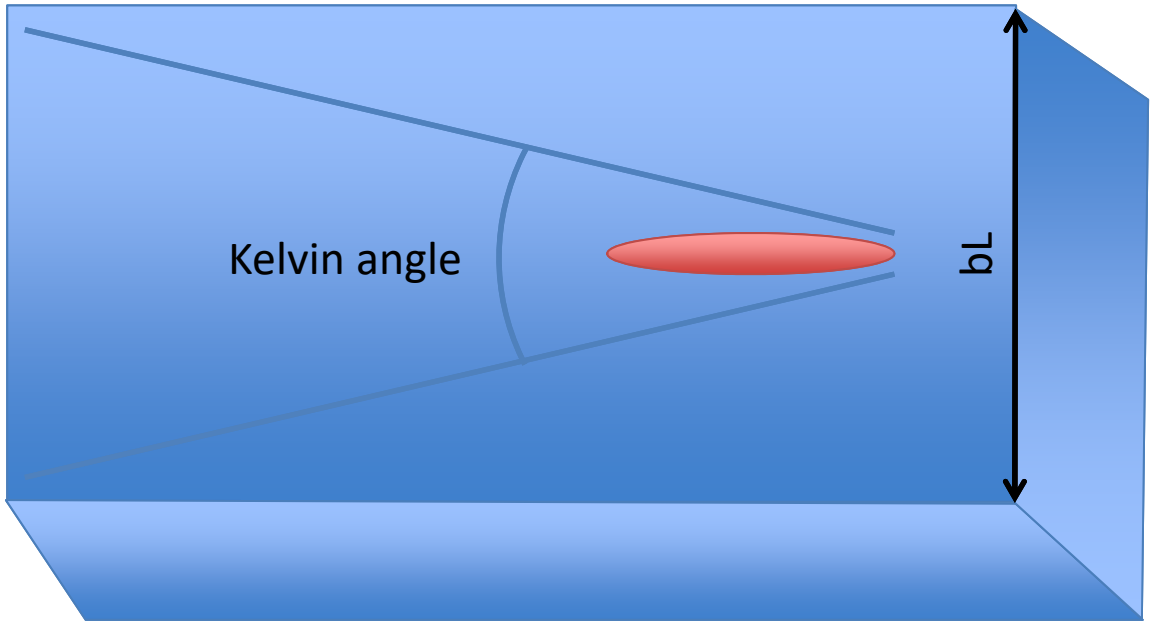
D must be set-up following the infinite depth condition for the wave theory: rigorously the depth must be greater than 1 wave length (Function of Speed)

A good choice is at least 1 ship length (for common ship speed).

Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions: Some empirical rules can be followed to correctly define a domain
   - The domain shouldn't influence the hull drag (different if shallow water condition or confined water conditions are taken in to account)

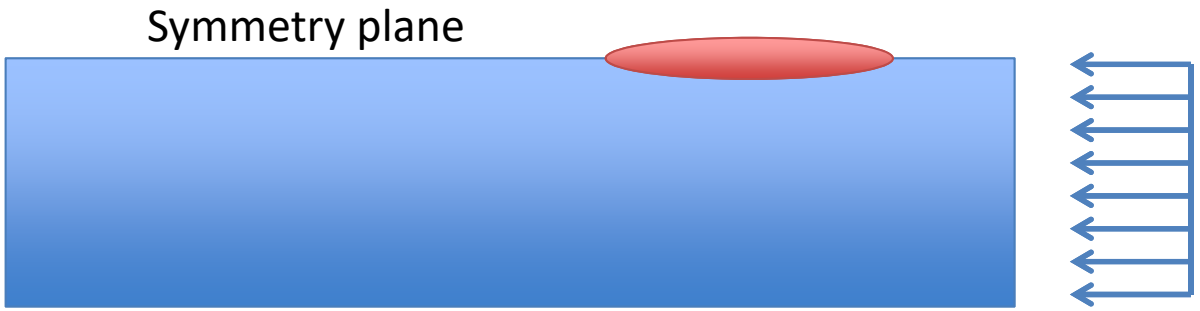


Side boundaries are set to include all the divergent waves, therefore the Kelvin angle (19.47°) can be used!

Hands-on session

Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions: Some empirical rules can be followed to correctly define a domain
   - The domain shouldn’t influence the hull drag (different if shallow water condition or confined water conditions are taken in to account)
   - The box shape is useful to set simple boundary conditions , moreover the ship symmetry can be used to reduce the mesh size



Hands-on session

Main steps to build a towing test simulation are:

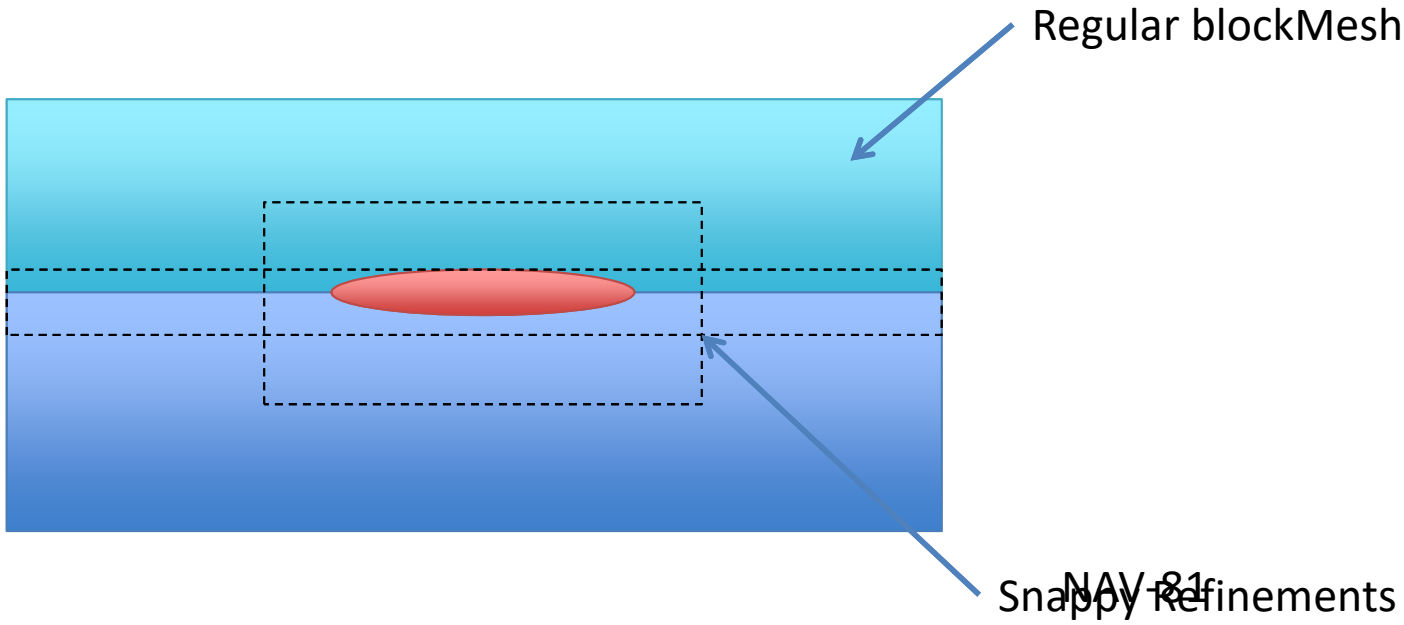
1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions : In OpenFOAM the domain has been set with the initial block mesh.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

```

Hands-on session

Main steps to build a towing test simulation are:

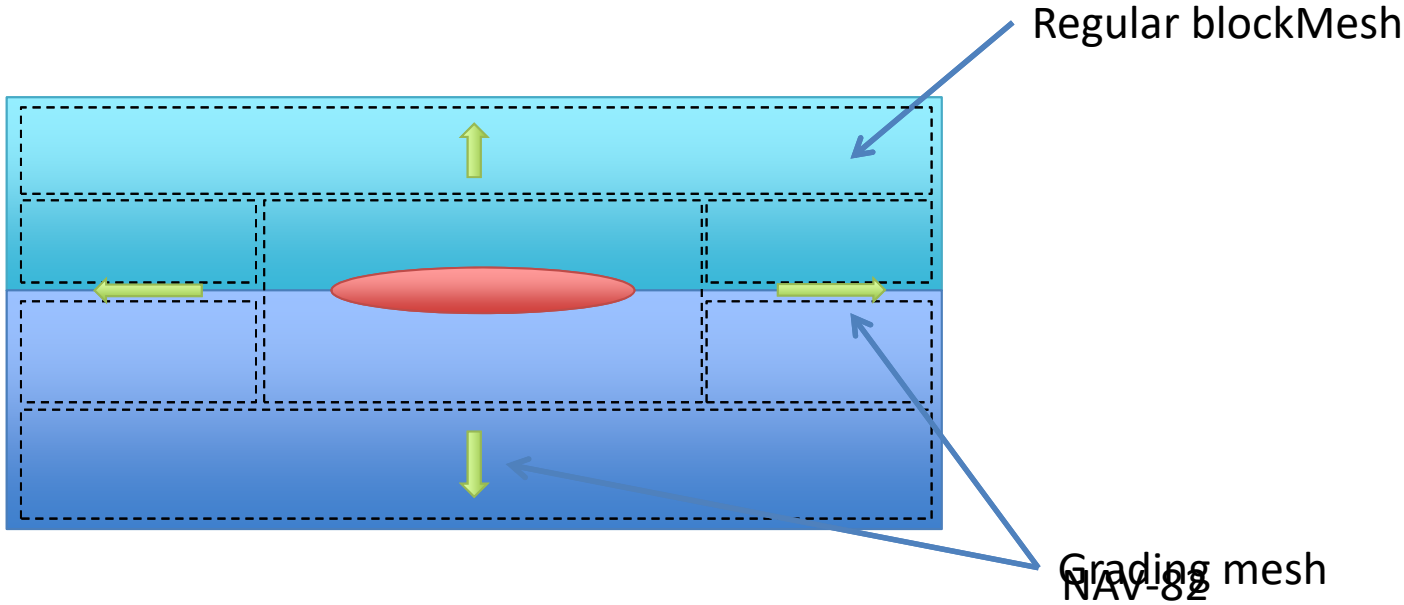
1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions
4. Create the Mesh: Different mesh strategies can be adopted for the ship problem (as seen in previous lectures). In this tutorial only SnapyHexMesh has been investigated.
    1. Create entire process with SnapyHexMesh



Hands-on session

Main steps to build a towing test simulation are:

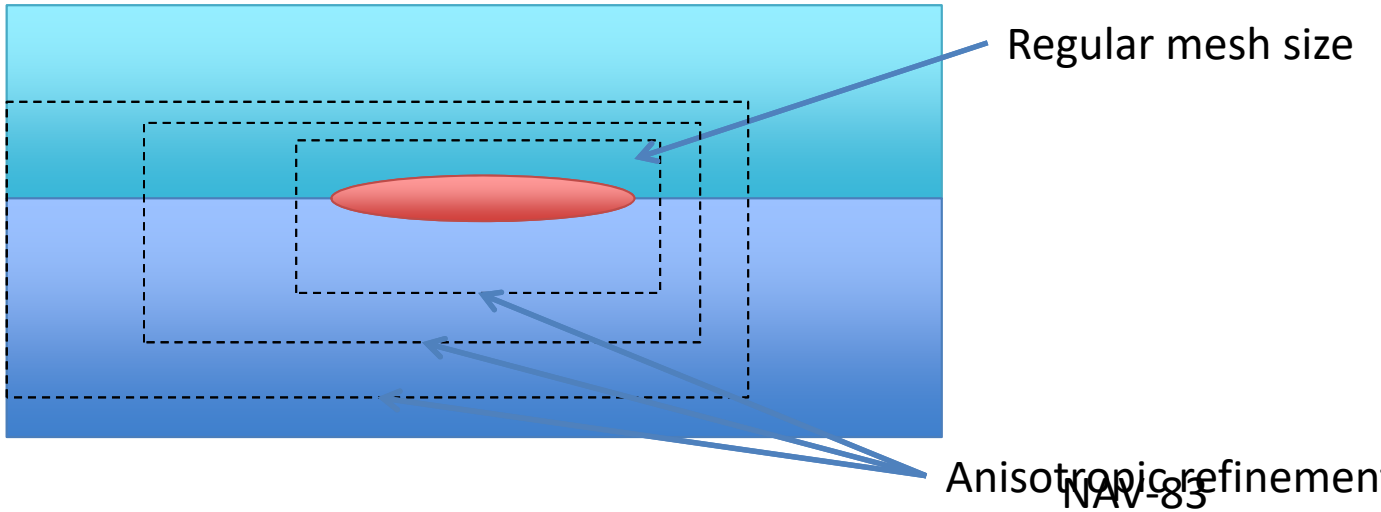
1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions
4. Create the Mesh: Different mesh strategies can be adopted for the ship problem (as seen in previous lectures). In this tutorial only SnapyHexMesh has been investigated.
    1. Create entire process with SnapyHexMesh
    2. Create multi blockMesh to include mesh grading



Hands-on session

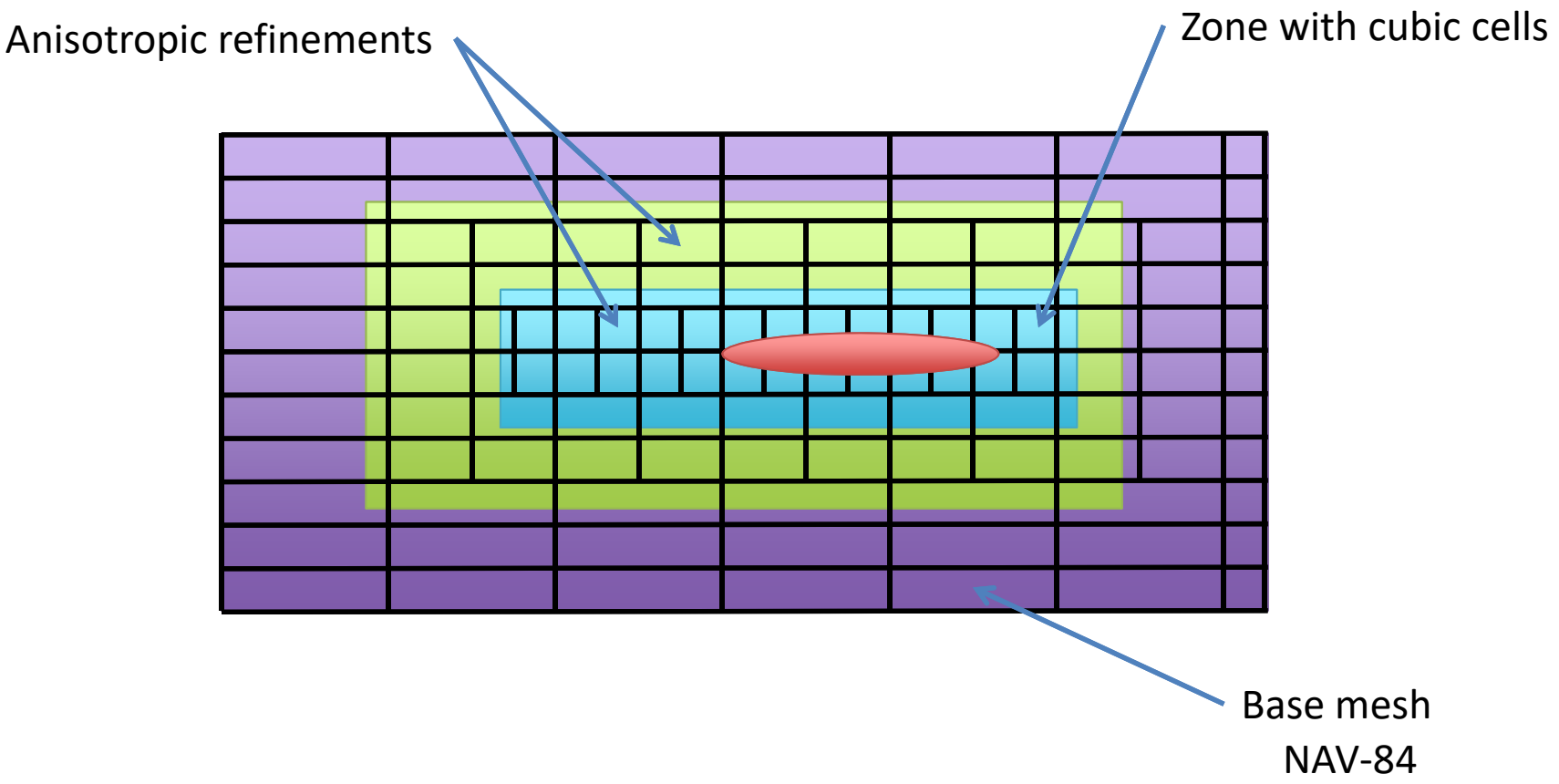
Main steps to build a towing test simulation are:

1. Create appropriate case directories with all the needed initial files
2. Create a usable .stl file
3. Define a domain shape and dimensions
4. Create the Mesh: Different mesh strategies can be adopted for the ship problem (as seen in previous lectures). In this tutorial only SnappyHexMesh has been investigated.
    1. Create entire process with SnappyHexMesh
    2. Create multi blockMesh to include mesh grading
    3. Use the mesh refinement to include anisotropic cells



Hands-on session

Mesh generation with SnappyHexMesh and mesh refinement: The main idea consists of createing a regular mesh (made only by Cubic cells) around the ship where snappyHexMesh will create the body-fitted mesh.



Hands-on session

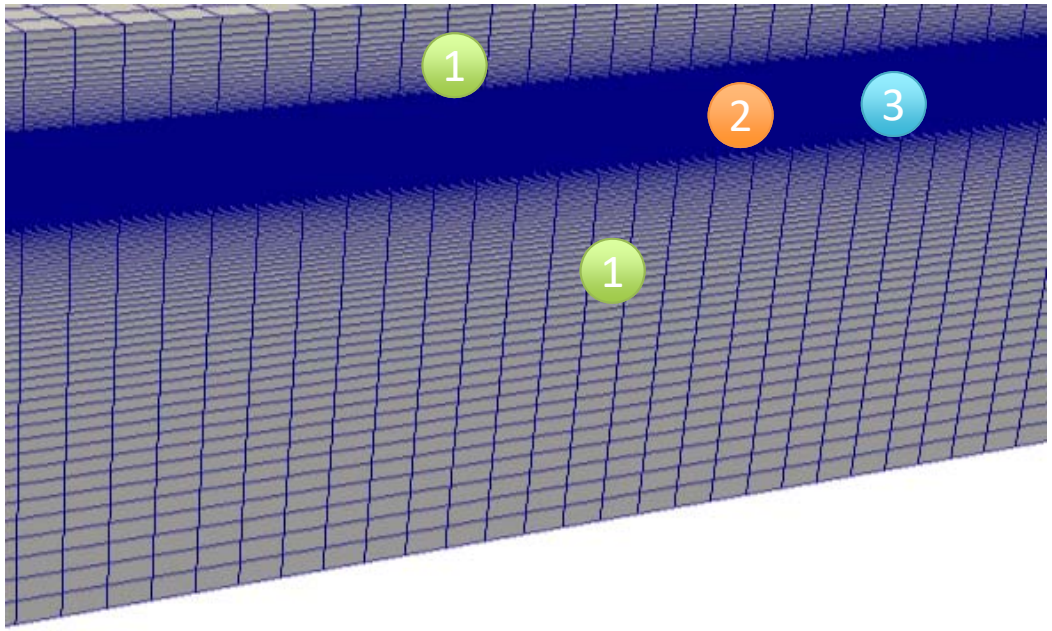
Mesh generation with SanppyHexMesh and mesh refinement:

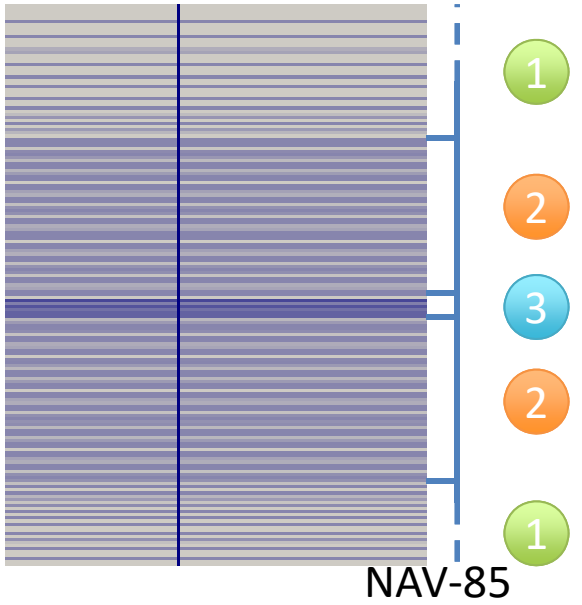
1. Start with ad-hoc blockMesh (under constant/polyMesh directory):

Zones 1 are used to increase the cell size in Z-direction far from the ship (grading mesh)

Zones 2 are used to create cubic celles near the hull

Zone 3 is used to refine the calm water free surface

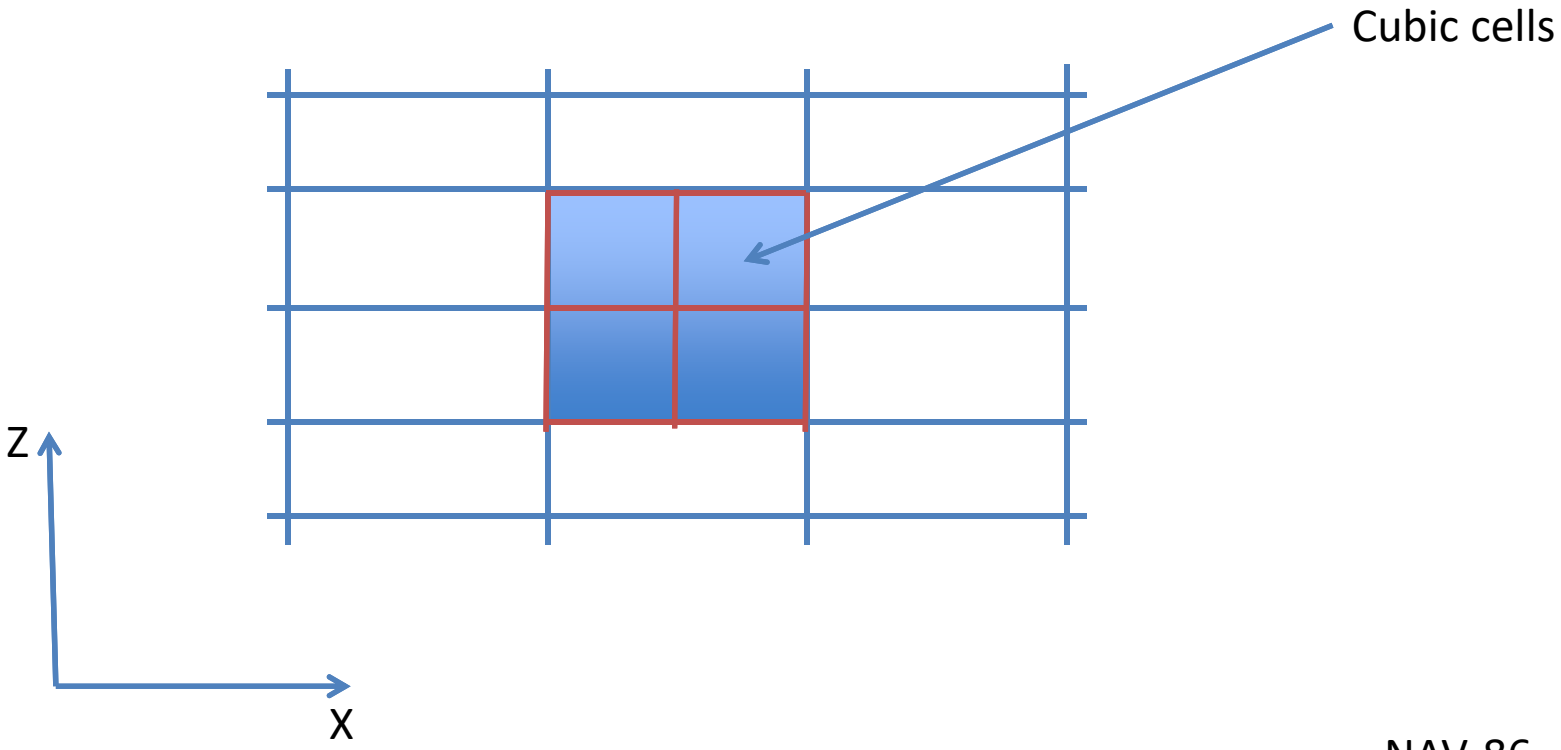




Hands-on session

Mesh generation with SnappyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.



Hands-on session

Mesh generation with SnappyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict): create a new cell region marked to be refined

```
actions(
    {
        name    c0;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (minX minY minZ) (maxX maxY maxZ);
        }
    }
);
```

Name of the region

Type of entity to be selected

Action to be done

Type of selection region

Data for the box…

Hands-on session

Mesh generation with SnappyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict):
    2. refineMesh (refineMeshDict): refine the mesh in x and y dierctions

```
set               c0;
coordinateSystem global;
globalCoeffs
{
    tan1            ( 1 0 0 );
    tan2            ( 0 1 0 );
}
directions      ( tan1 tan2 );
useHexTopology  no;
geometricCut    yes;
writeMesh       no;
```

Name of set that will be refined

Definition of the directions global/local

Direction to be refined X and Y

Split only hex cells…no good in this case!

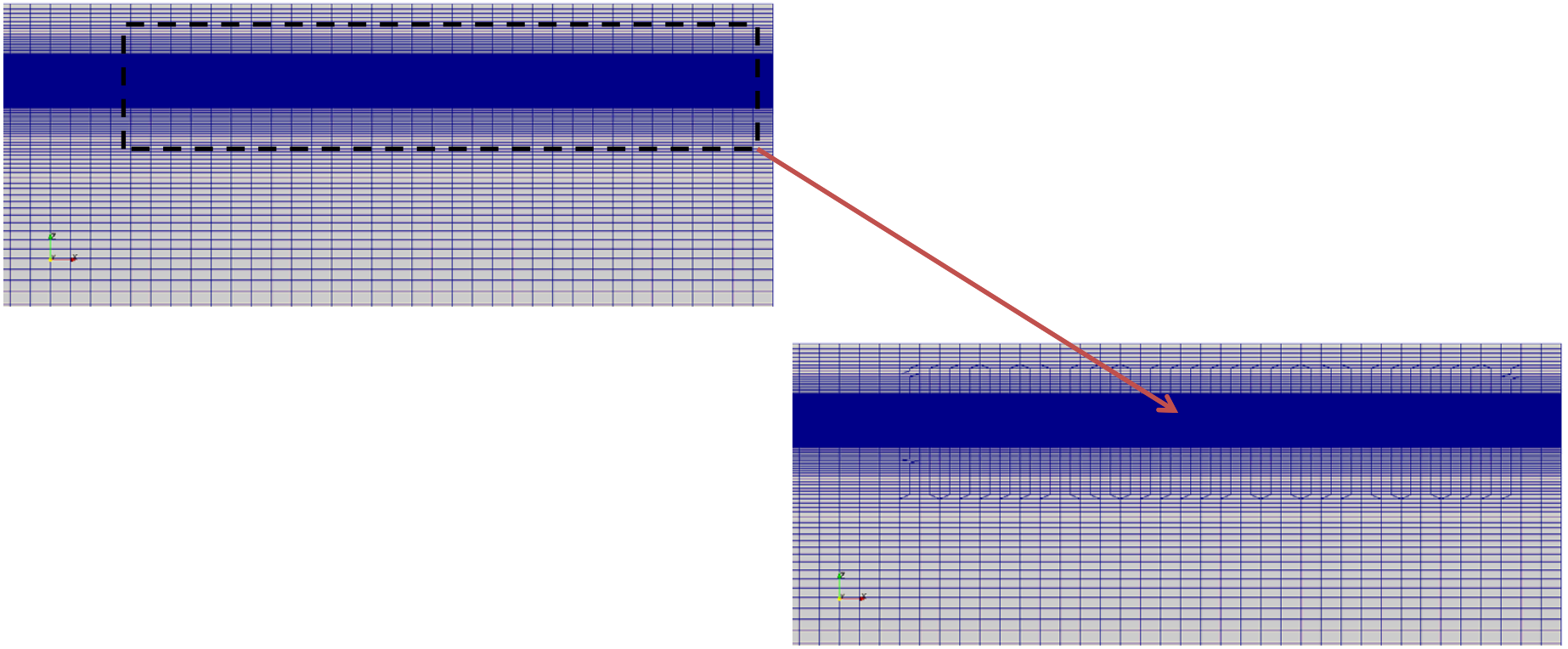
Take in to account topological aspects

Write intermediate mesh

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

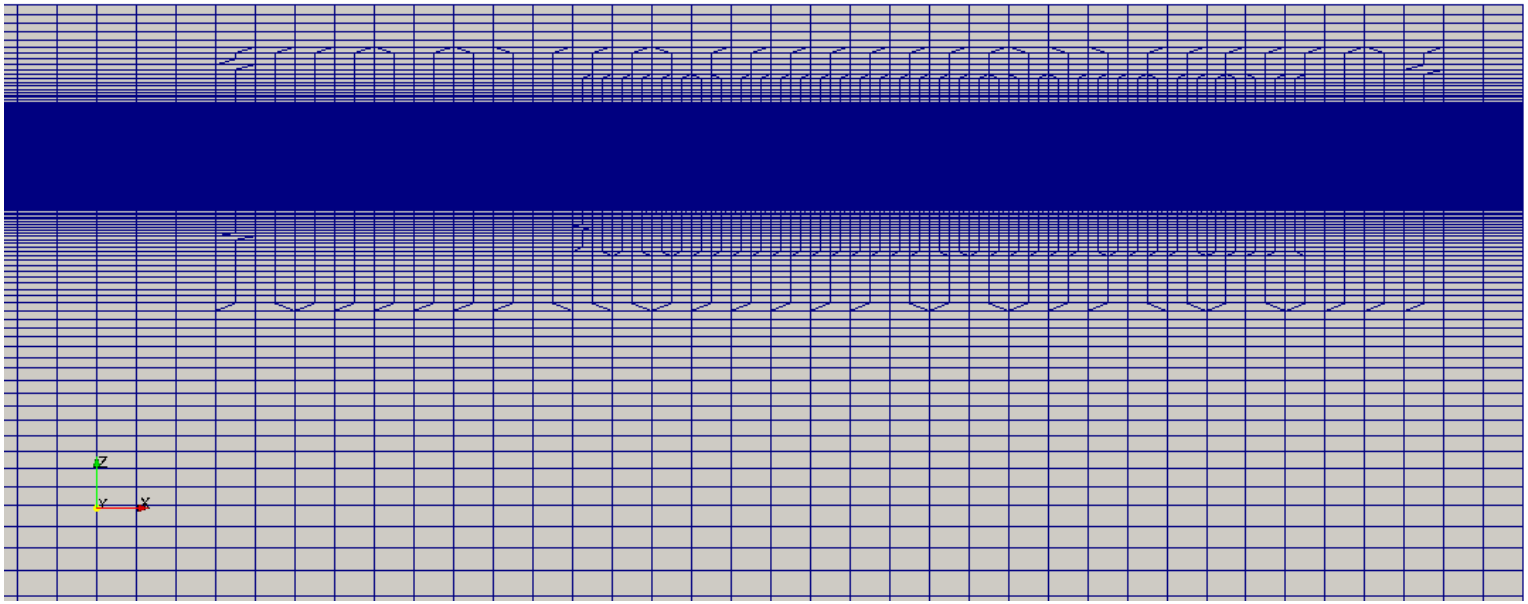
1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict):
    2. refineMesh (refineMeshDict): repeat the process till the ship mesh is cubic



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

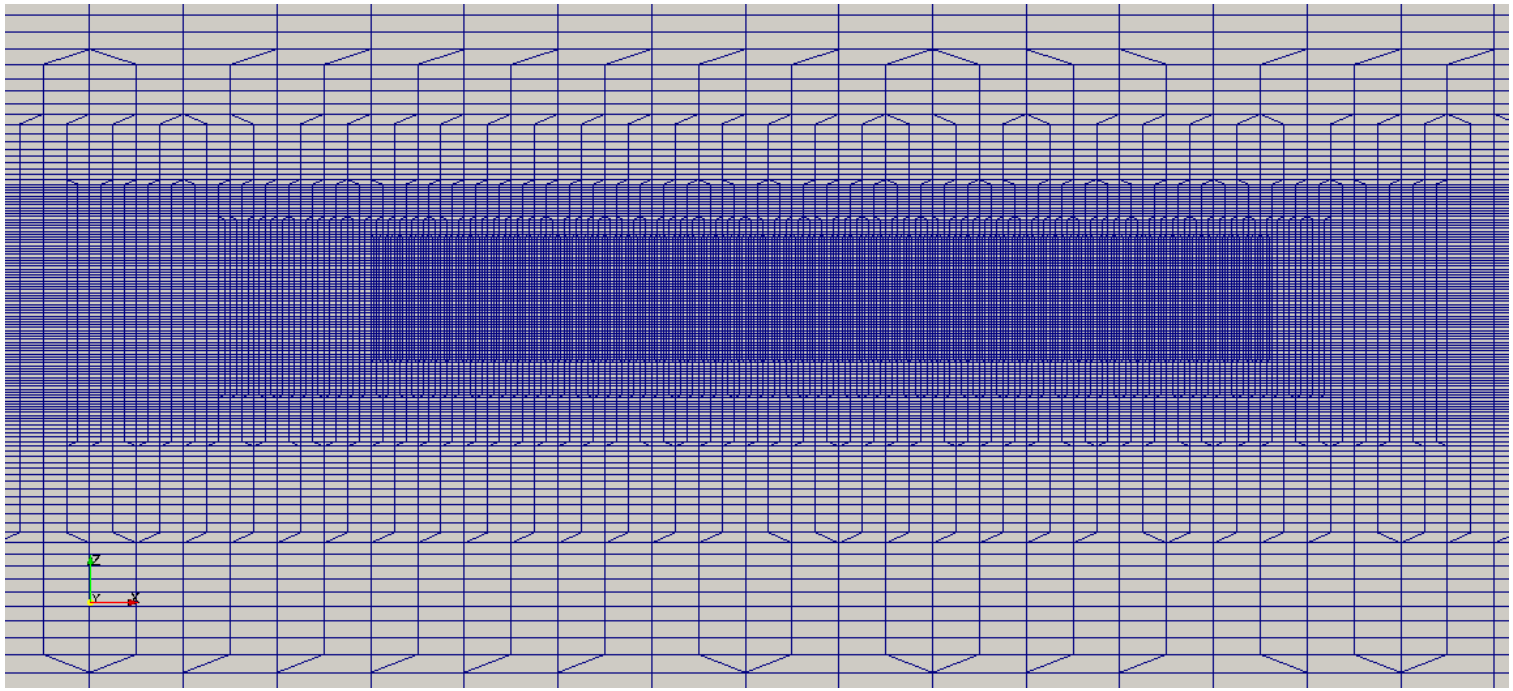
1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict):
    2. refineMesh (refineMeshDict): repeat the process till the ship mesh is cubic



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

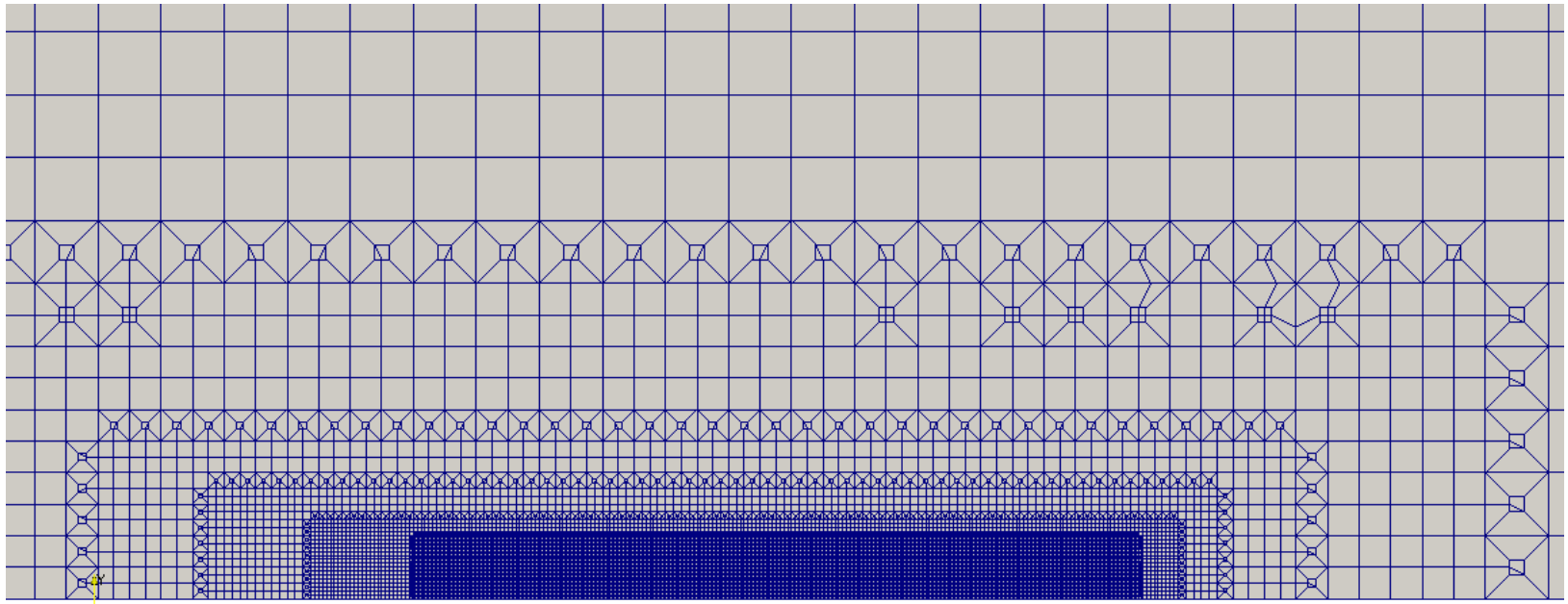
1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict):
    2. refineMesh (refineMeshDict): repeat the process till the ship mesh is cubic



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh only in the X and Y directions to create regular cubic cell around the ship or the zone on which the snappy will work.
    1. topoSet (topoSetDict):
    2. refineMesh (refineMeshDict): repeat the process till the ship mesh is cubic



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull: The snappyHexMesh command uses a snappyHexMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      autoHexMeshDict;
}

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

// Which of the steps to run
//castellatedMesh false;
//snap            false;

castellatedMesh true;
snap            true;
addLayers       true;

```

Create the appropriate refinement and identify the internal mesh

Snap the boundary points on the real surface

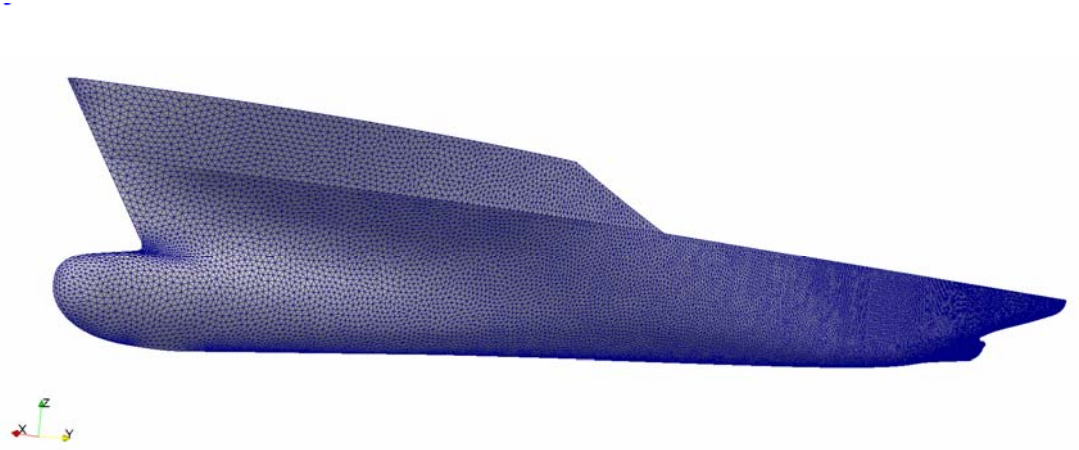
Add the prism layer mesh to solve the boundary layer

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry: include the .stl file as a triSurfMesh, one for each wall boundary needed (for simplicity only one boundary)

```
33 geometry
34 {
35 CarenaKCS2.stl
36     {
37         type triSurfaceMesh;
38         name Carena;
39
40         patchInfo
41         {
42             type wall;
43         }
44     }
45 };
```



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh

```
50 castellatedMeshControls
51 {
52     maxLocalCells 100000;
53     maxGlobalCells 15000000;
54     minRefinementCells 2;
55     nCellsBetweenLevels 4;
56
57     features
58     (
59     );
60
61     refinementSurfaces
62     {
63         Carena
64         {
65             level (1 1);
66         }
67     }
68
69     resolveFeatureAngle 60;
70
71     refinementRegions
72     {
73     }
74
75     locationInMesh (-4 2 0.001);
76     allowFreeStandingZoneFaces true;
77 }
```

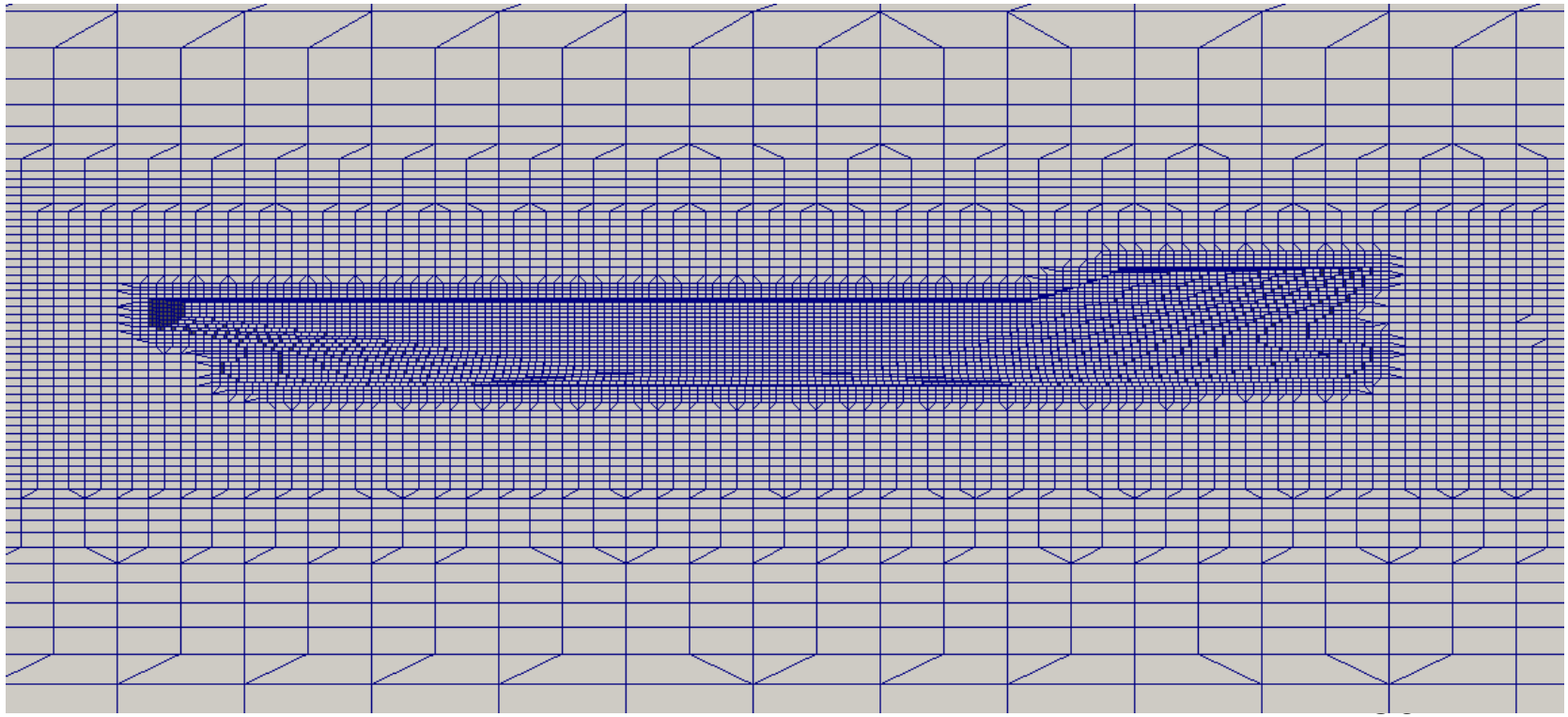
Set the boundary name to refine the surface

Pay attention to put the point inside the mesh and outside the hull

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

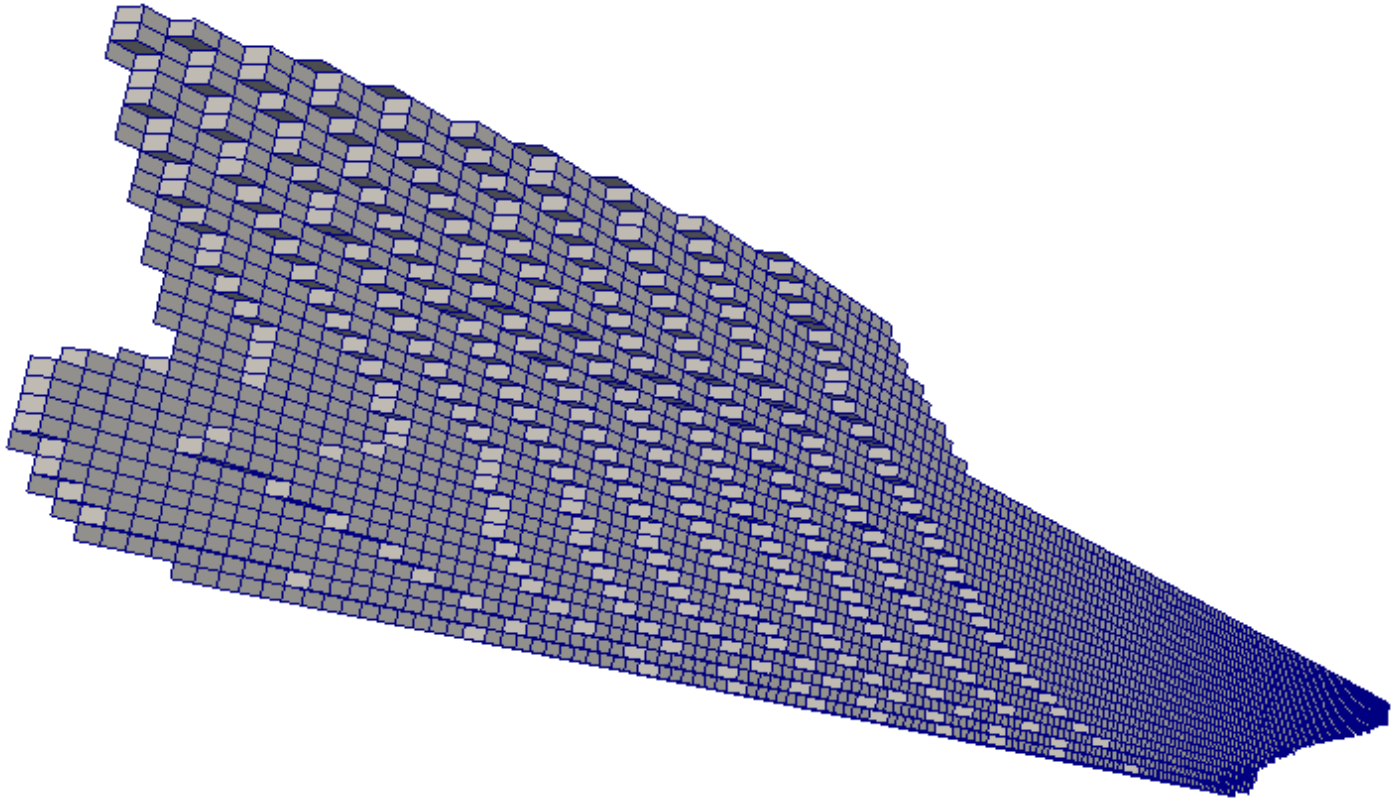
1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh

```
160 // Settings for the snapping.
161 snapControls
162 {
163     //- Number of patch smoothing iterations before finding correspondence
164     //  to surface
165     nSmoothPatch 10;
166
167     //- Relative distance for points to be attracted by surface feature point
168     //  or edge. True distance is this factor times local
169     //  maximum edge length.
170 //    tolerance 4.0;
171         tolerance 4.0;
172
173     //- Number of mesh displacement relaxation iterations.
174     nSolveIter 400;
175
176     //- Maximum number of snapping relaxation iterations. Should stop
177     //  before upon reaching a correct mesh.
178         nRelaxIter 10;
179         // Feature snapping
180
181         // Number of feature edge snapping iterations.
182         // Leave out altogether to disable.
183         nFeatureSnapIter 10;
184
185         // Detect (geometric only) features by sampling the surface
186         // (default=false).
187         implicitFeatureSnap false;
188 }
189
```

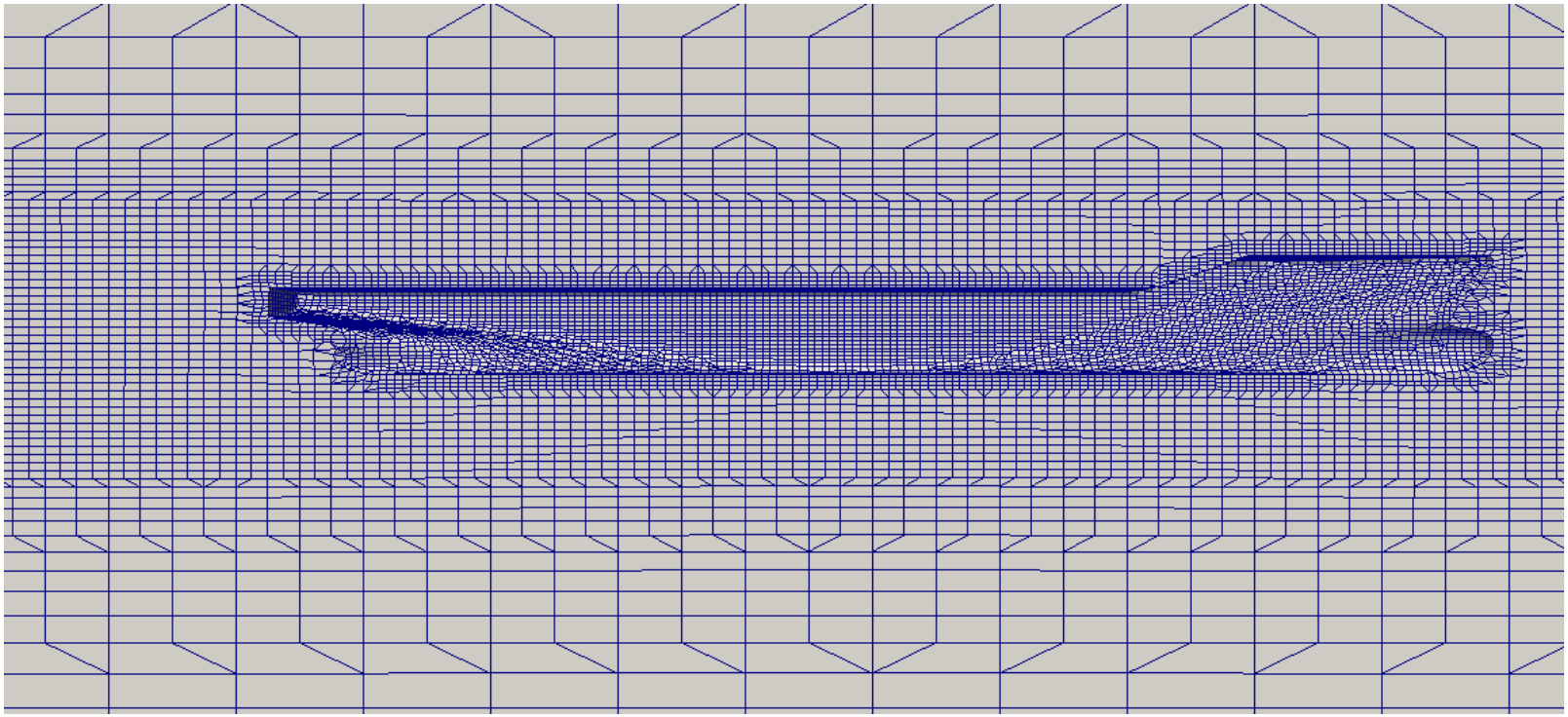
Better snapping result

How many times the deformations can be reduced to have a correct mesh

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

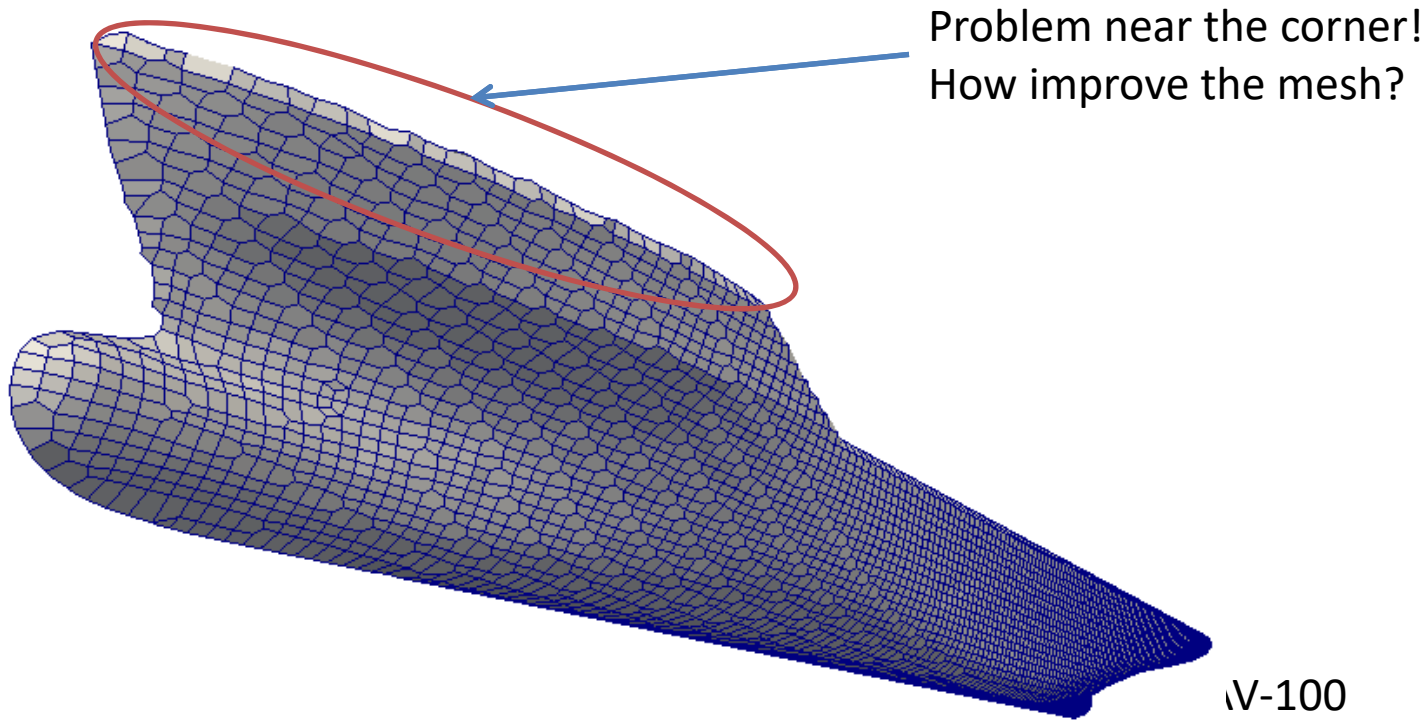
1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh

```
160 // Settings for the snapping.
161 snapControls
162 {
163     //- Number of patch smoothing iterations before finding correspondence
164     //  to surface
165     nSmoothPatch 10;
166
167     //- Relative distance for points to be attracted by surface feature point
168     //  or edge. True distance is this factor times local
169     //  maximum edge length.
170 //    tolerance 4.0;
171         tolerance 4.0;
172
173     //- Number of mesh displacement relaxation iterations.
174     nSolveIter 400;
175
176     //- Maximum number of snapping relaxation iterations. Should stop
177     //  before upon reaching a correct mesh.
178         nRelaxIter 10;
179         // Feature snapping
180
181         // Number of feature edge snapping iterations.
182         // Leave out altogether to disable.
183         nFeatureSnapIter 10;
184
```

```
// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
    {
        file "CarenaEdge.eMesh";
        level 2;
    }
);
```

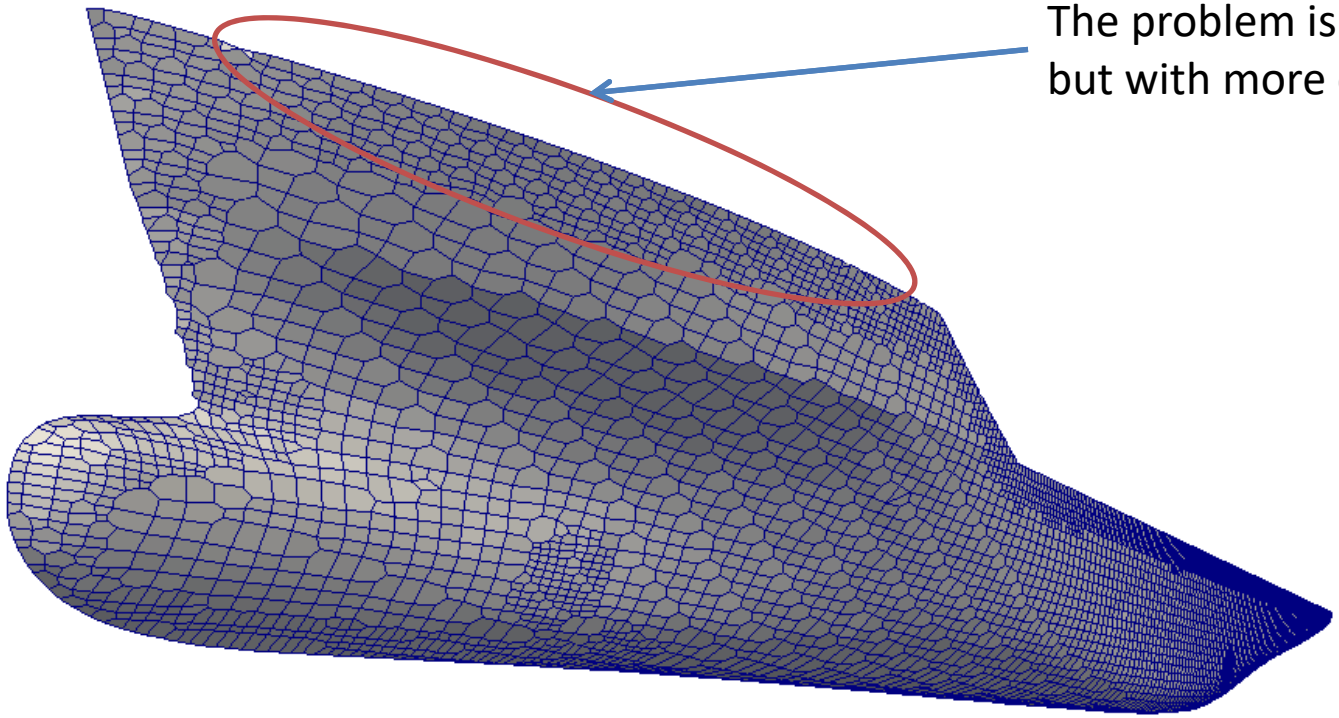
Set all true and add the Explicit feature

```
// Detect (geometric only) features by sampling the surface
// (default=false).
implicitFeatureSnap true;

//- Use castellatedMeshControls::features (default = true)
explicitFeatureSnap true;

//- Detect points on multiple surfaces (only for explicitFeatureSnap)
multiRegionFeatureSnap false;
```

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh



Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh
    4. Add a prismatic layer

Relative or Absolute thickness

Global or Local parameters

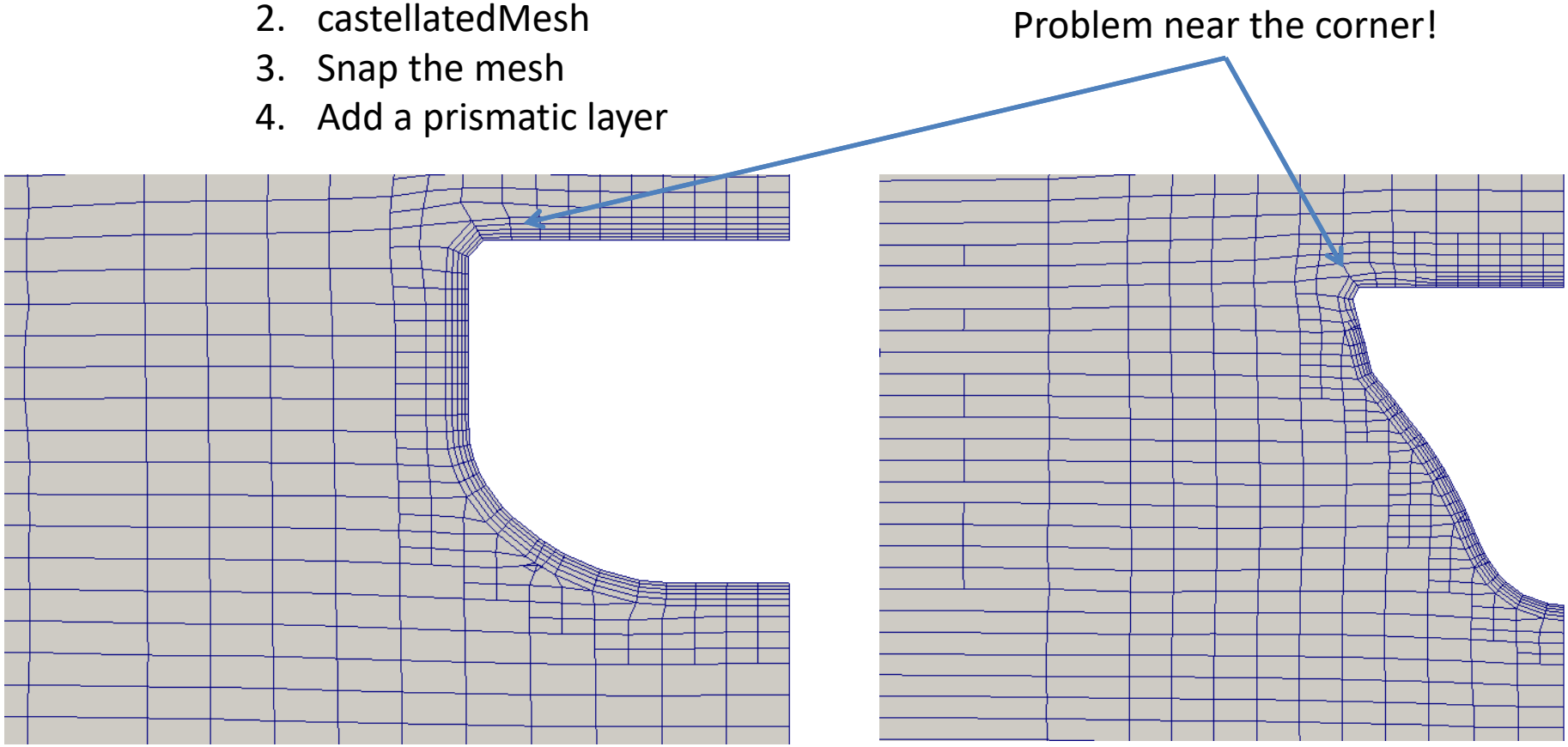
Other important parameters to create the mesh

```
192 // Settings for the layer addition.
193 addLayersControls
194 {
195     relativeSizes false;
196
197     layers
198     {
199         Carena
200         {
201             nSurfaceLayers 5;
202         }
203     }
204     expansionRatio 1.2;
205     finalLayerThickness 0.01;
206     minThickness 0.0001;
207     nGrow 0;
208
209     // Advanced settings
210     featureAngle 170;
211     nRelaxIter 5;
212     nSmoothSurfaceNormals 2;
213     nSmoothNormals 3;
214     nSmoothThickness 5;
215     maxFaceThicknessRatio 0.5;
216     maxThicknessToMedialRatio 0.3;
217     minMedianAxisAngle 90;
218     nBufferCellsNoExtrude 0;
219     nLayerIter 50;
220     nRelaxedIter 20;
221 }
222
```

Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh
    4. Add a prismatic layer

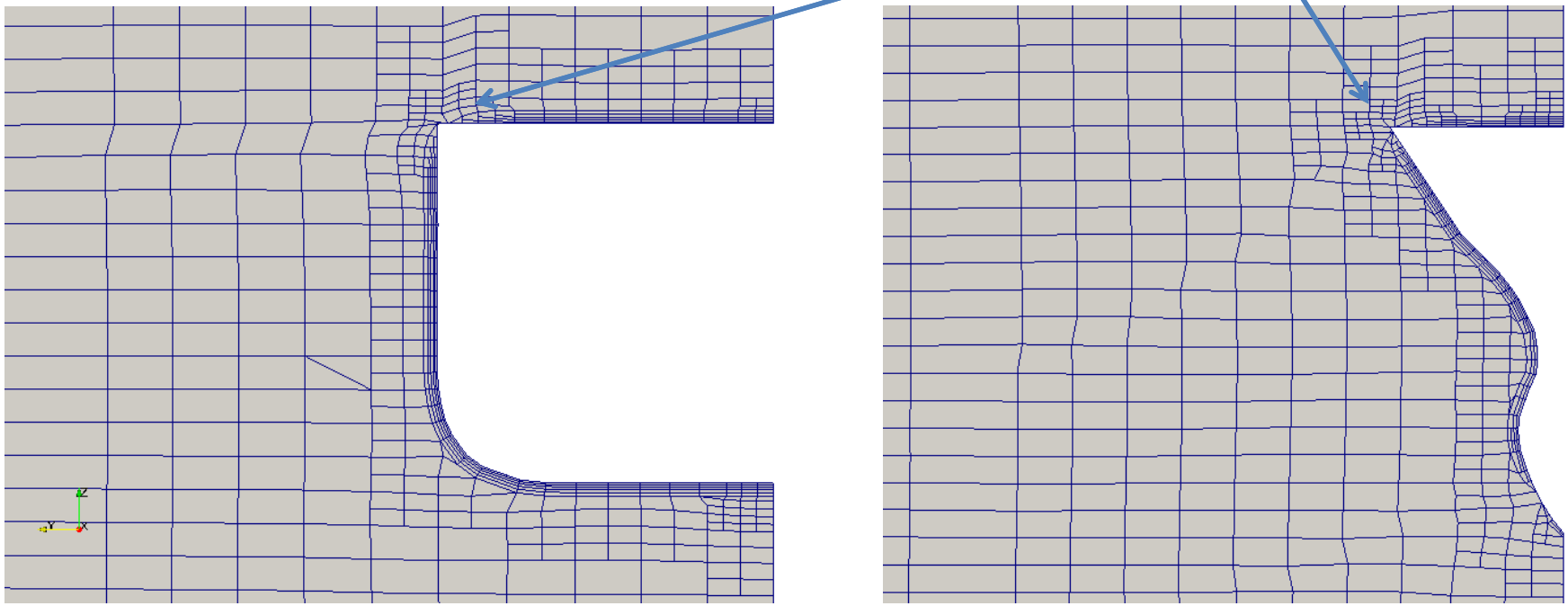


Hands-on session

Mesh generation with SanppyHexMesh and mesh refinement:

1. Start with ad-hoc blockMesh
2. Refine the mesh
3. Create the real mesh around the hull:
    1. geometry
    2. castellatedMesh
    3. Snap the mesh
    4. Add a prismatic layer

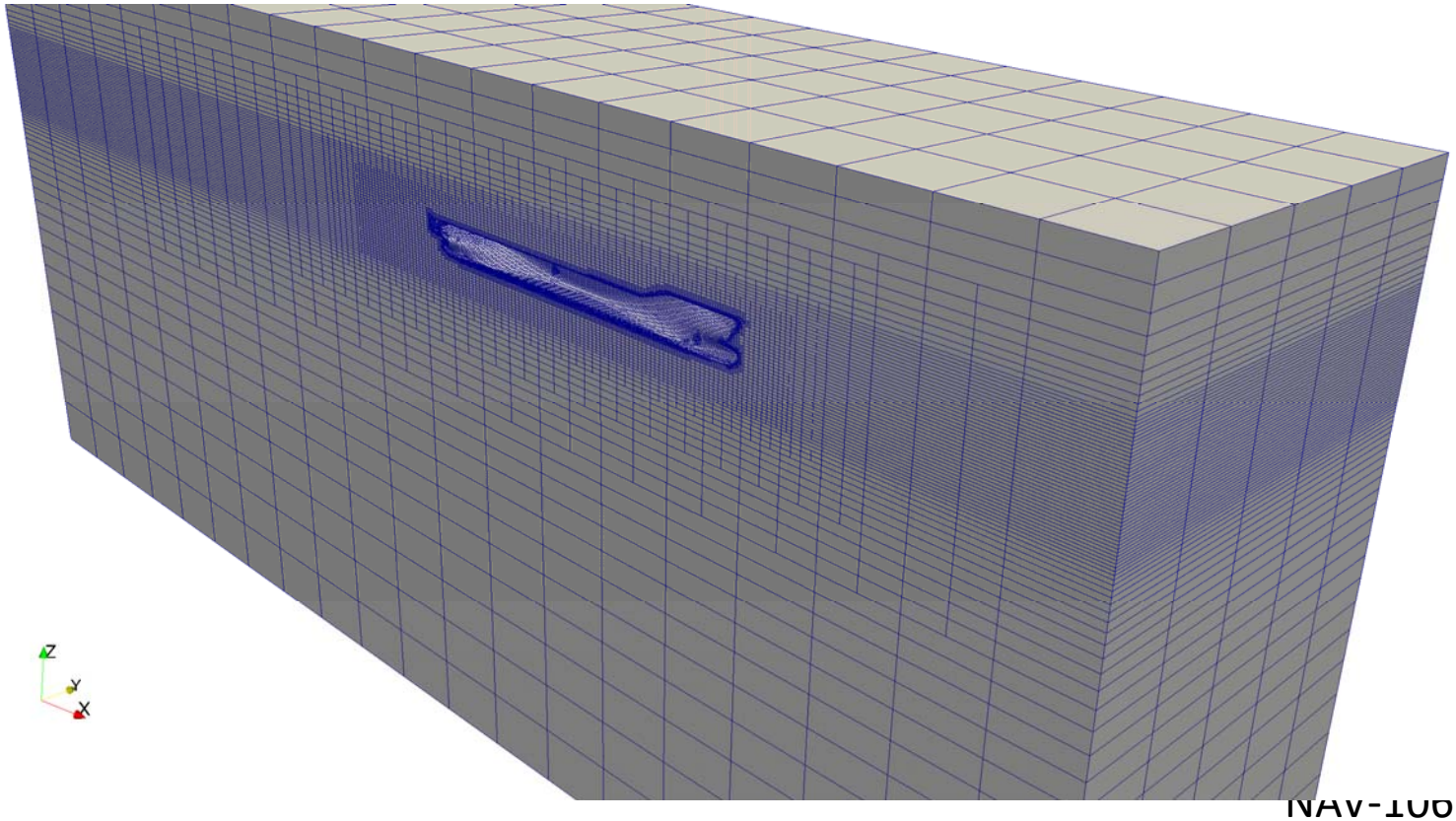
Other problem near the corner!



Hands-on session

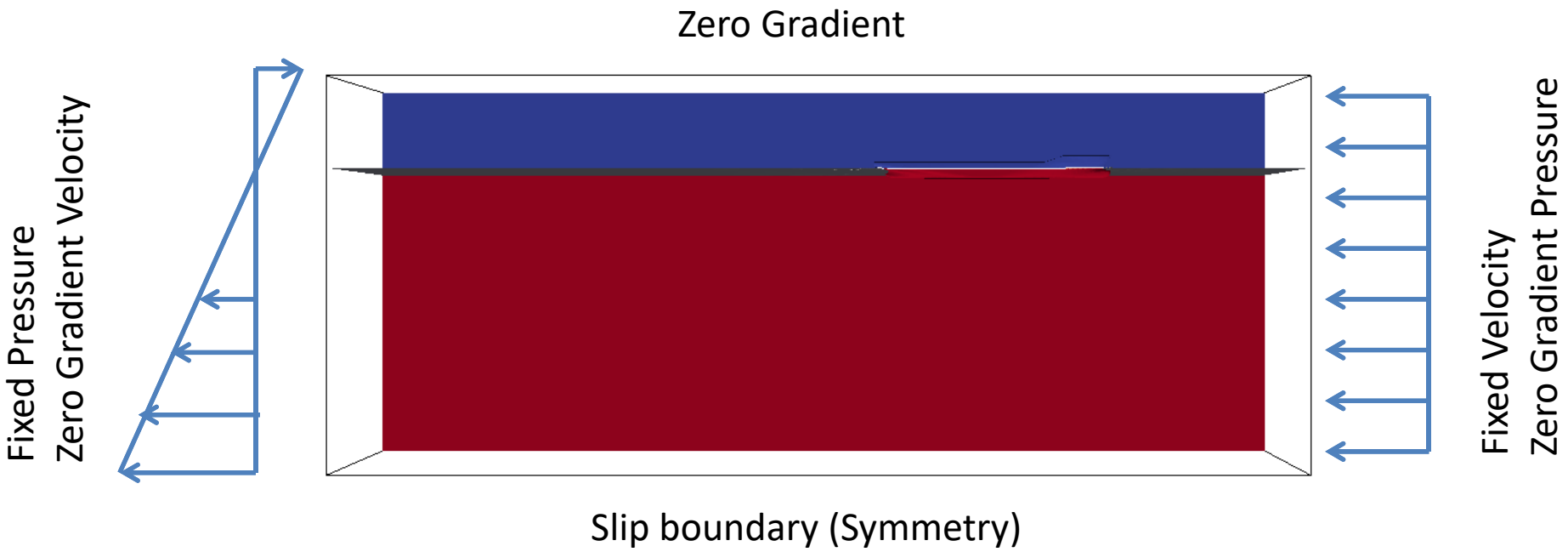
Mesh generation with SanppyHexMesh and mesh refinement:

To have a good mesh quality is almost “impossible”!
But a good enough mesh is possible! (hard work)



Hands-on session

Set initial fields and boundary conditions: How to set a correct initial field in OpenFOAM?



Hands-on session

Set initial fields and boundary conditions: How to set a correct initial field in OpenFOAM?

The *setFields* utility can be useful

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-100 -100 -100) (100 100 0.0);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
    boxToFace
    {
        box (-100 -100 -100) (100 100 0.0);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);

```

Default value for the variable

Region type to set cells and faces

Value of the variable inside of the region

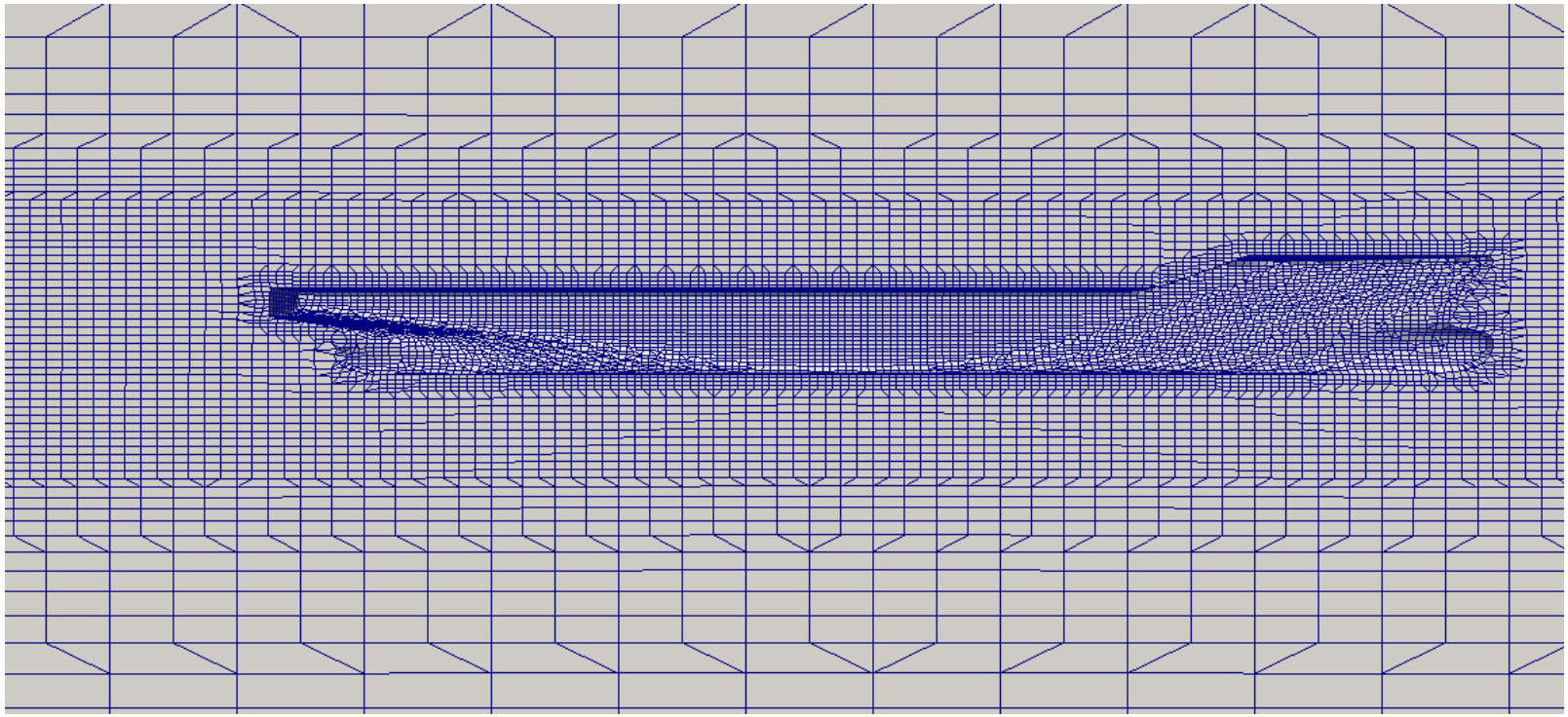
Hands-on session*

InterFoam is the OF solver to perform a steady solution.

Hands-on session

How take into account the trim effects on the ship drag?

An unsteady solver should be used, moreover the Dynamic Mesh feature should be used. -> interDyMFoam



Hands-on session

How take into account the trim effects on the ship drag?

An unsteady solver should be used, moreover the Dynamic Mesh feature should be used. -> interDyMFoam

Inclusion of the constant/dynamicMeshDict

List of moving boundary

Distance on which the morphing take place

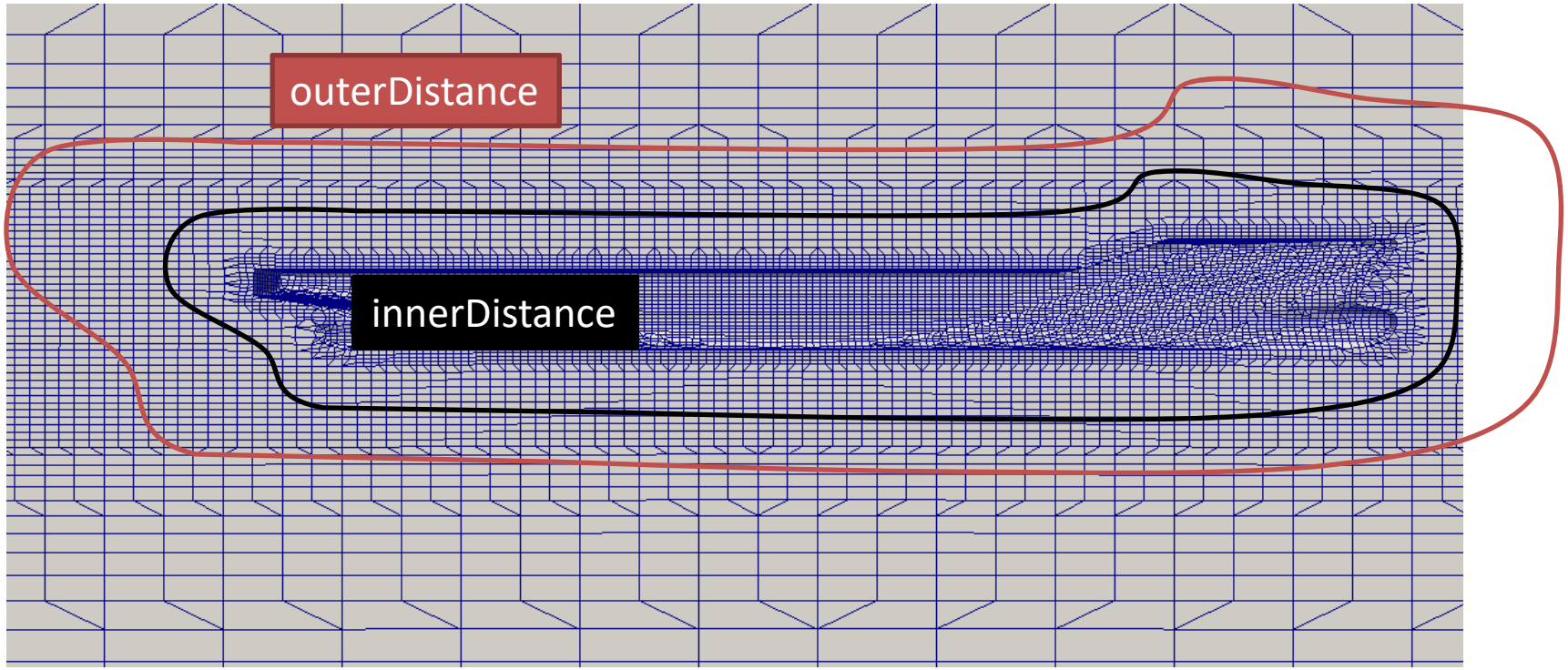
Geometry mass proprieties

Constraints and Restraints

```
17 dynamicFvMesh       dynamicMotionSolverFvMesh;
18 motionSolverLibs    ("libsixDoFRigidBodyMotion.so");
19 solver              sixDoFRigidBodyMotion;
20 sixDoFRigidBodyMotionCoeffs
21 {
22     patches         ( Carena );
23     innerDistance   0.3;
24     outerDistance   1;
25
26     centreOfMass    (3.5854 0 0.11145);
27     mass            824.5;
28     momentOfInertia (2500 2500 2500);
29     rhoInf          1000;
30     report          on;
31     accelerationRelaxation 0.3;
32     value           uniform (0 0 0);
33
34     constraints{
35         zAxis
36         {
37             sixDoFRigidBodyMotionConstraint line;
38             direction     (0 0 1);
39         }
40         yPlane
41         {
42             sixDoFRigidBodyMotionConstraint axis;
43             axis          (0 1 0);
44         }
45     }
46     restraints{
47         translationDamper
48         {
49             sixDoFRigidBodyMotionRestraint linearDamper;
50             coeff         8596;
51         }
52         rotationDamper
53         {
54             sixDoFRigidBodyMotionRestraint sphericalAngularDamper;
55             coeff         11586;
56         }
57     }
58 }
```

Hands-on session

Distance on which the morphing take place



Hands-on session

How take in to account the trim effects on the ship drag?

An unsteady solver should be used, moreover the Dynamic Mesh feature should be used. -> interDyMFoam

Inclusion of the constant/dynamicMeshDict

Inclidion of an adding Boundary condition
0/pointDisplacement

Set *fixed* for each fixed boundary

Set *calculated* for each moving boundary

```
17
18 dimensions      [0 1 0 0 0 0 0];
19
20 internalField   uniform (0 0 0);
21
22 boundaryField
23 {
24     //- Set patchGroups for constraint patches
25     //#include "${WM_PROJECT_DIR}/etc/caseDicts/setConstraintTypes"
26
27     inlet
28     {
29         type            fixedValue;
30         value           uniform (0 0 0);
31     }
32
33     outlet
34     {
35         type            fixedValue;
36         value           uniform (0 0 0);
37     }
38
39     atmosphere
40     {
41         type            fixedValue;
42         value           uniform (0 0 0);
43     }
44     "Sym.*"
45     {
46         type            symmetryPlane;
47     }
48     Carena
49     {
50         type            calculated;
51     }
52 }
53
54
```

Hands-on session*

InterDyMFoam is the OF solver to perform an unsteady solution with moving mesh.

Today's lecture

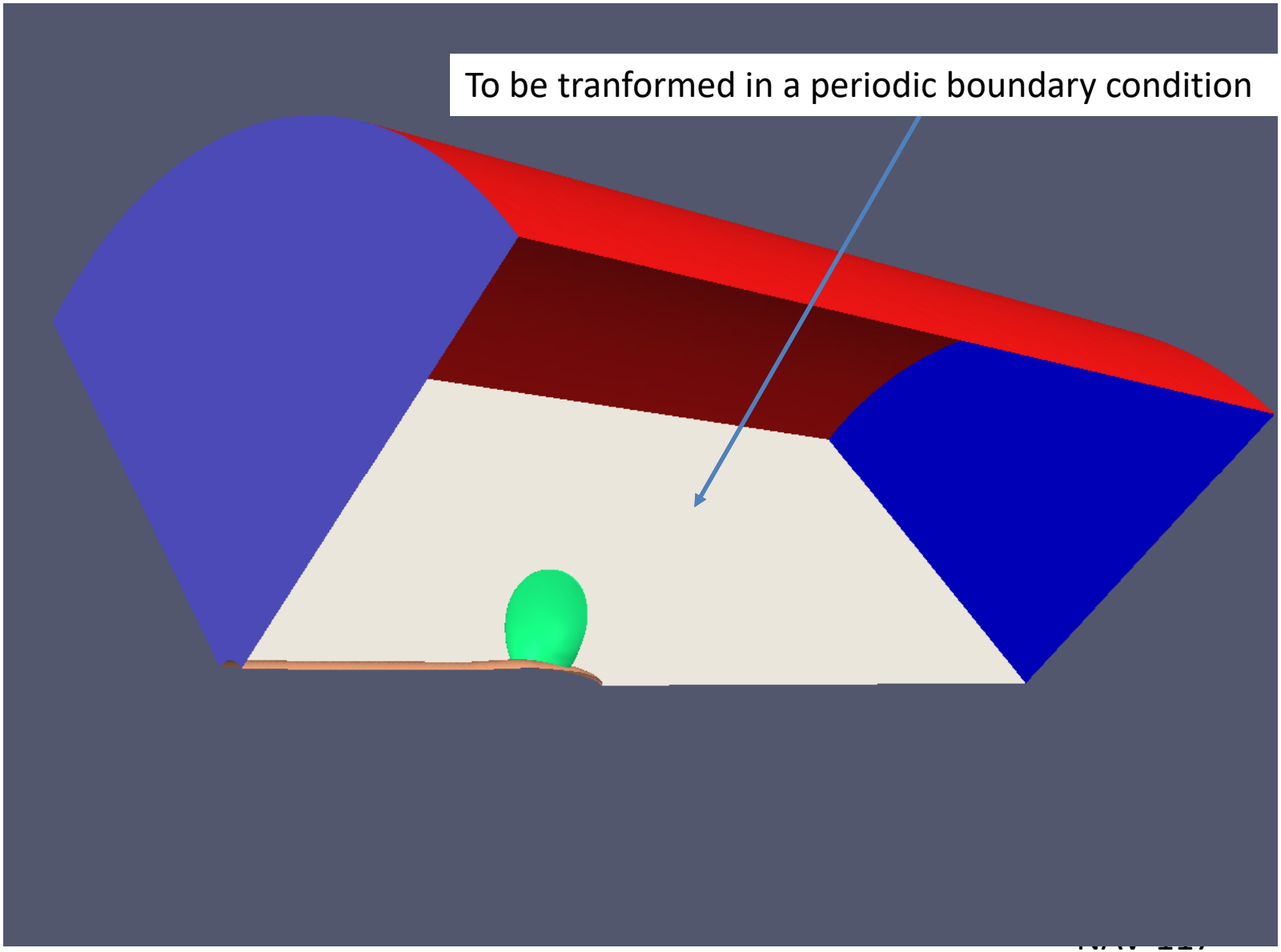
1. Introduction to the hydrodynamic naval problems
    1. Ship Resistance
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

Hands-on session

E779A – A propeller Test Case:

- Open Water Calculations,
- MRF approach with a domain reduced to an angular sector (360°/number of blades) to save cells and computational time,
- Need of periodic/cyclic boundary conditions to account for the “axial periodocity”

Hands-on session



Hands-on session

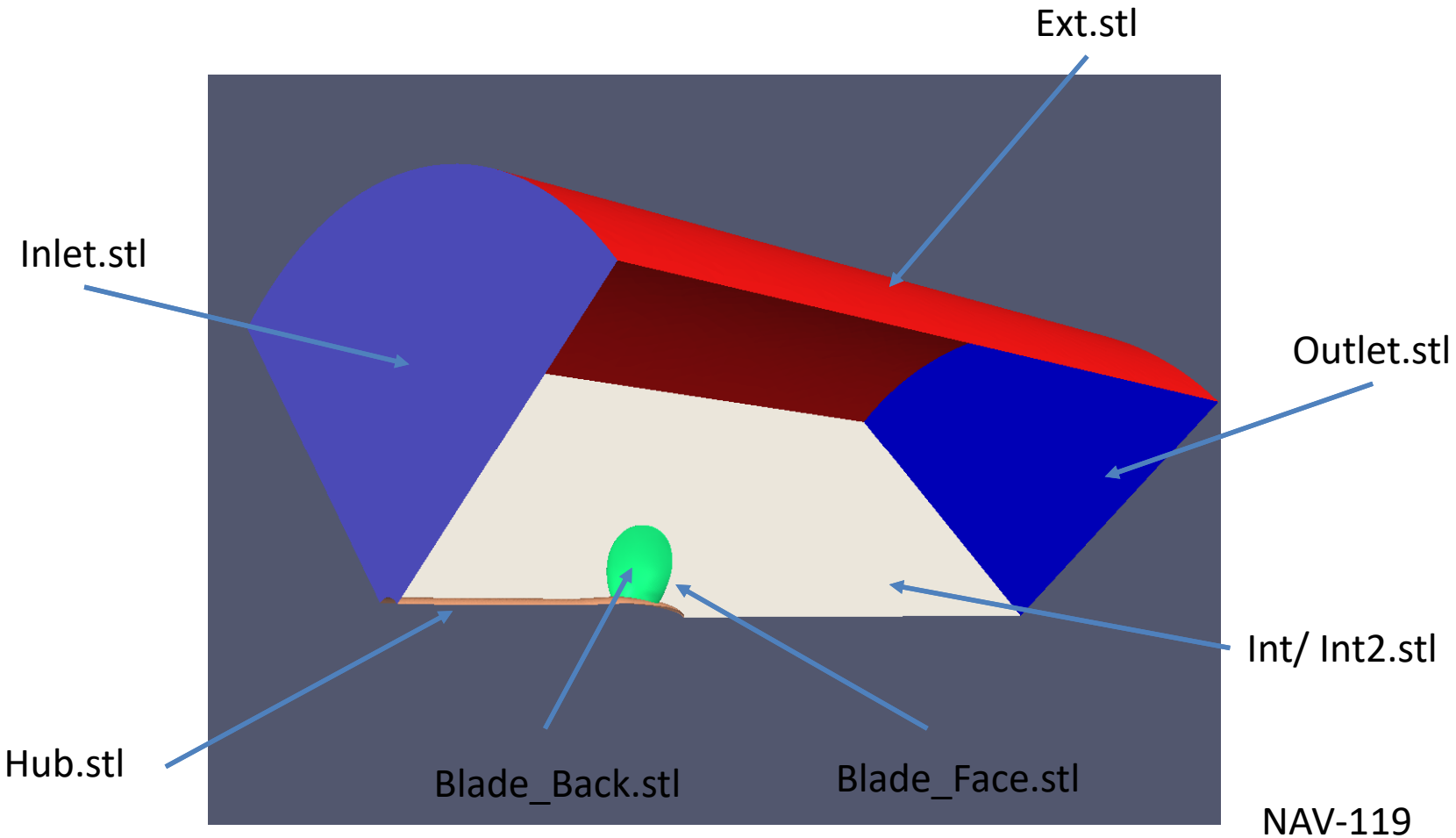
E779A – A propeller Test Case:

- Angular domain definition with `snappyHexMesh`
- Angular domain definition with `blockMesh+snappyHexMesh`

1. .STL files in the `constant/triSurface` dir.
2. Each patch separately defined in order to better include blade Leading Edge – Trailing Edge and Hub fillets

Hands-on session

- Domain definition with `snappyHexMesh`, separated STL patches.



Hands-on session

- Extraction of feature edges:
  - With `surfaceFeatureExtract:`

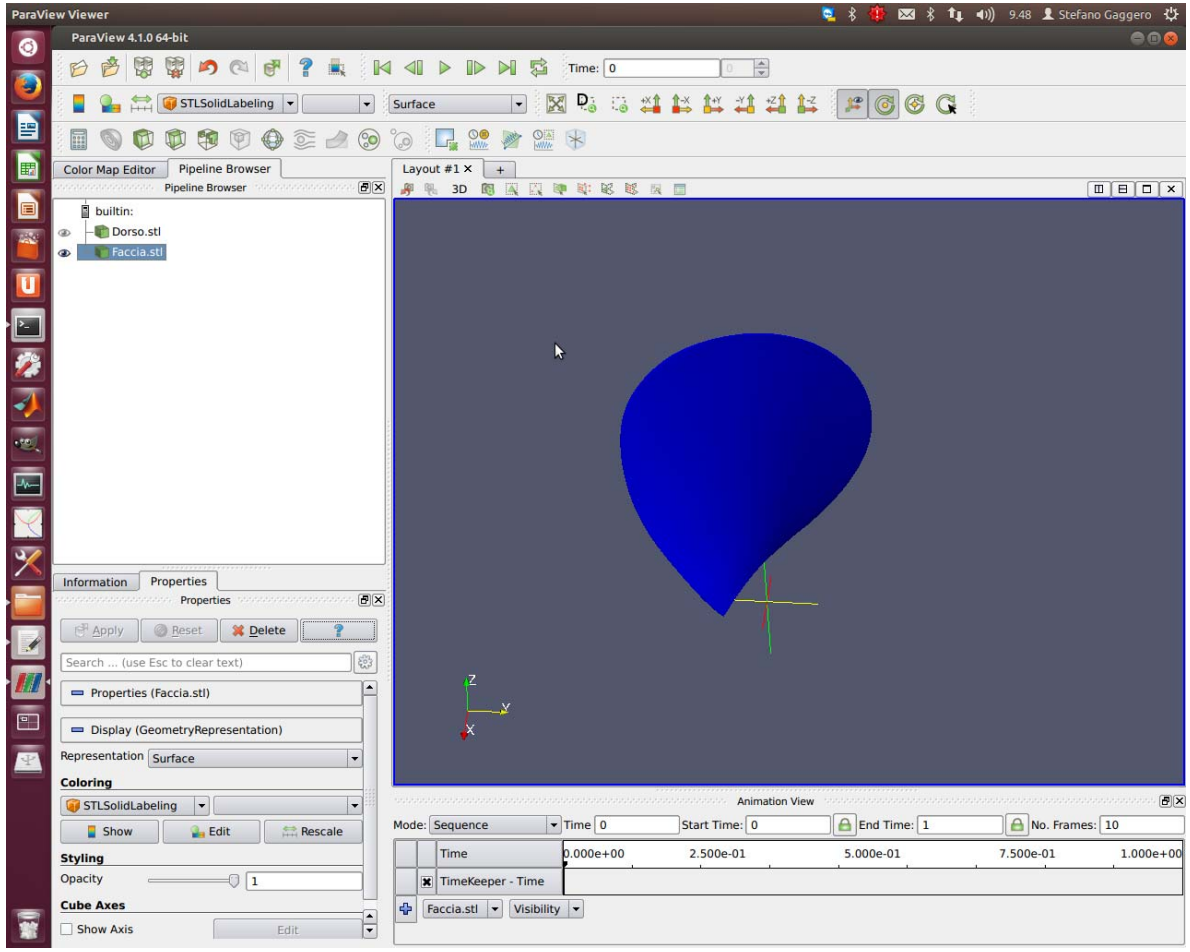
```
Hub.stl
{
    // How to obtain raw features (extractFromFile || extractFromSurface)
    extractionMethod    extractFromSurface;

    extractFromSurfaceCoeffs
    {
        // Mark edges whose adjacent surface normals are at an angle less
        // than includedAngle as features
        // - 0   : selects no edges
        // - 180: selects all edges

        includedAngle   160;
    }

    // Keep open edges (edges with 1 connected face)
        openEdges       yes;

        // Write features to obj format for postprocessing
        writeObj                yes;
}
```

When STL patches are separated, openEdges are always included in eMesh files for mesh refinements and feature edges snappyng….

…define STL patches in accordance to edges you want to snap to!

Hands-on session

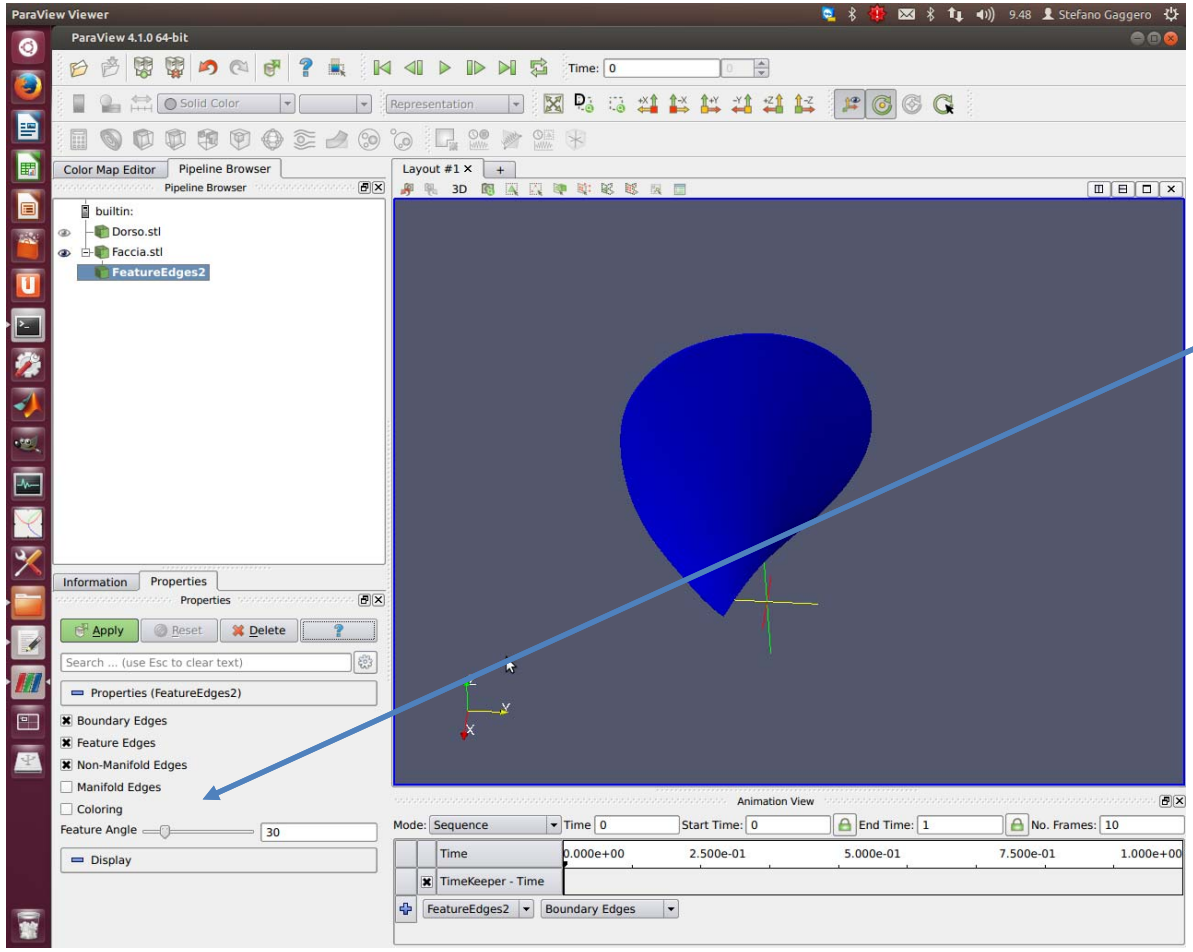
- Extraction of feature edges:
  - With `paraView:`



- Open STL file

Hands-on session

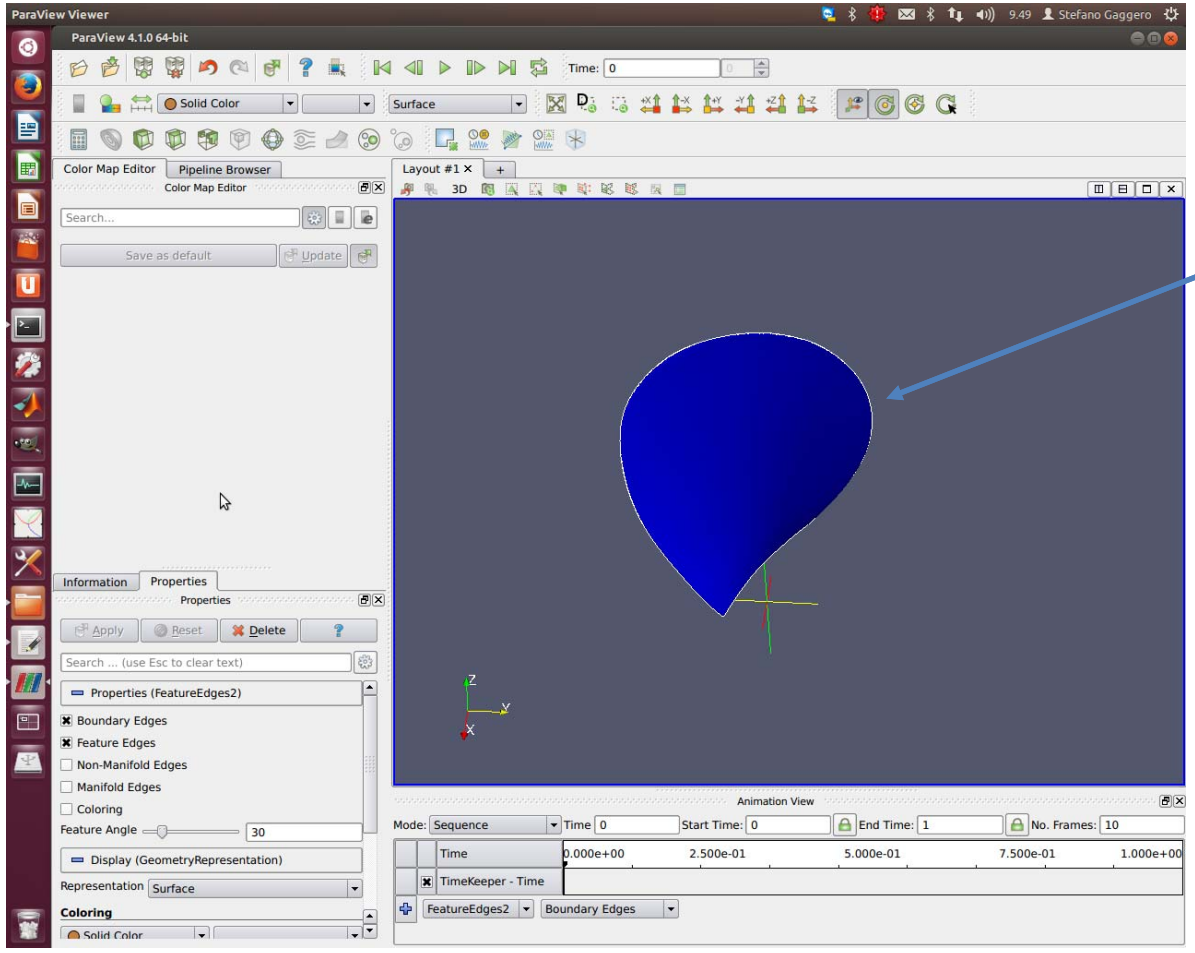
- Extraction of feature edges:
  - With `paraView:`



- Open STL file
- Use the filter *FeatureEdge* and custom parameters

Hands-on session

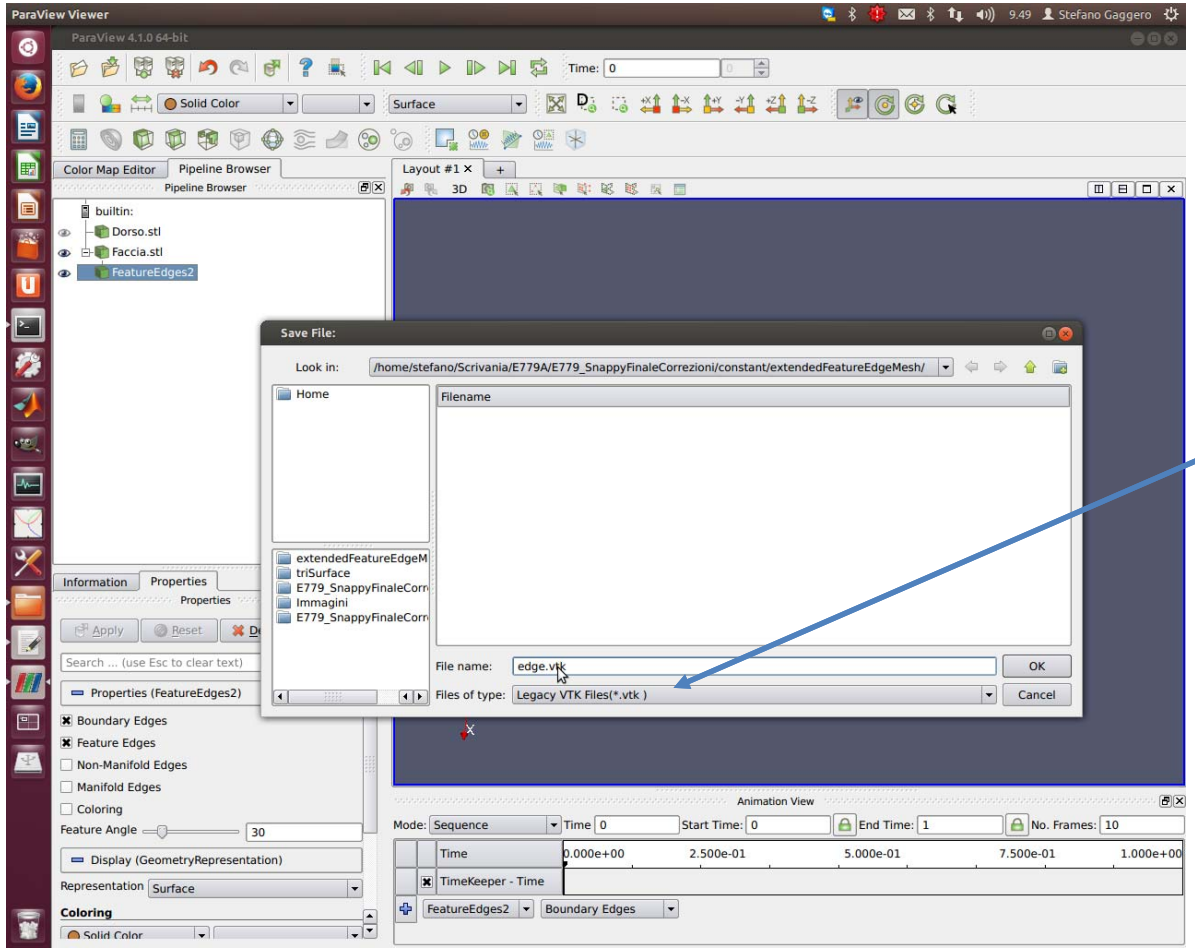
- Extraction of feature edges:
  - With `paraView:`



- Open STL file
- Use the filter *FeatureEdge* and custom parameters
- Feature Curves (including Open Edges) evidenced

Hands-on session

- Extraction of feature edges:
  - With `paraView:`



- Open STL file
- Use the filter *FeatureEdge* and custom parameters
- Feature Curves (including Open Edges) evidenced
- Export evidenced Feature Lines by «Save Data» command (Use .vtk file format)

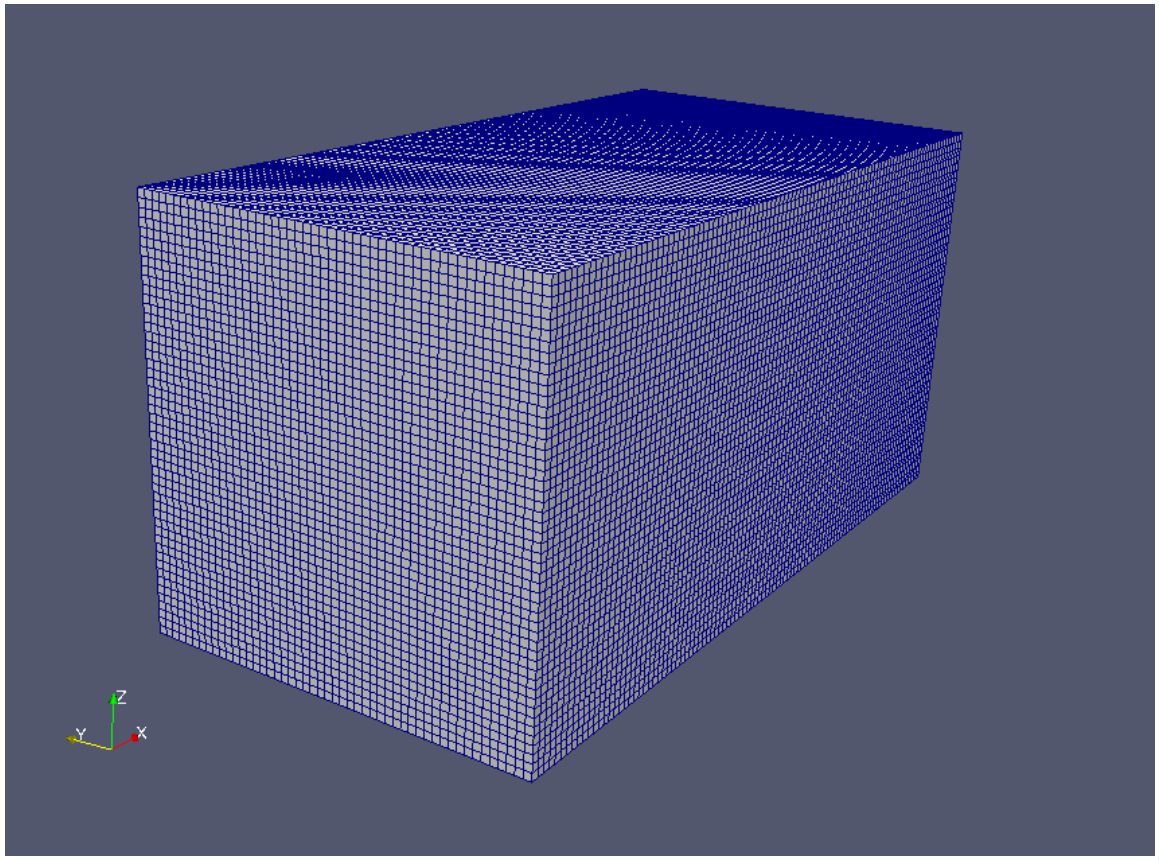
Hands-on session

- Extraction of feature edges:
  - With `paraView:`
  - Convert extracted feature line (.vtk) in .eMesh format by using:

```
surfaceFeatureConvert file.vtk file.eMesh
```

Hands-on session

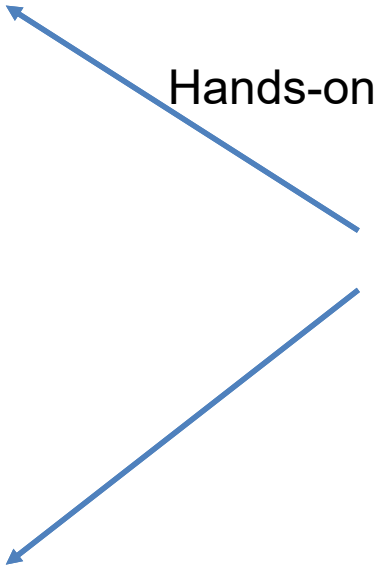
- As usual with `snappyHexMesh`:
- Define, with `blockMesh`, an initial domain of cubic cells,



Hands-on session

- As usual with `snappyHexMesh`:
- Each STL treated separately

```
geometry
{

    Hub.stl
    {
        type triSurfaceMesh;
        name Hub;
    }
    Blade_Face.stl
    {
        type triSurfaceMesh;
        name Blade_Face;
    }
    Blade_Back.stl
    {
        type triSurfaceMesh;
        name Blade_Back;
    }
    Ext.stl
    {
        type triSurfaceMesh;
        name Ext;
    }
    Int.stl
    {
        type triSurfaceMesh;
        name Int;
    }
    Int2.stl
    {
```



Hands-on session

All the STL identified with a name (the same will be visualized in paraview)

Hands-on session

- As usual with `snappyHexMesh`:

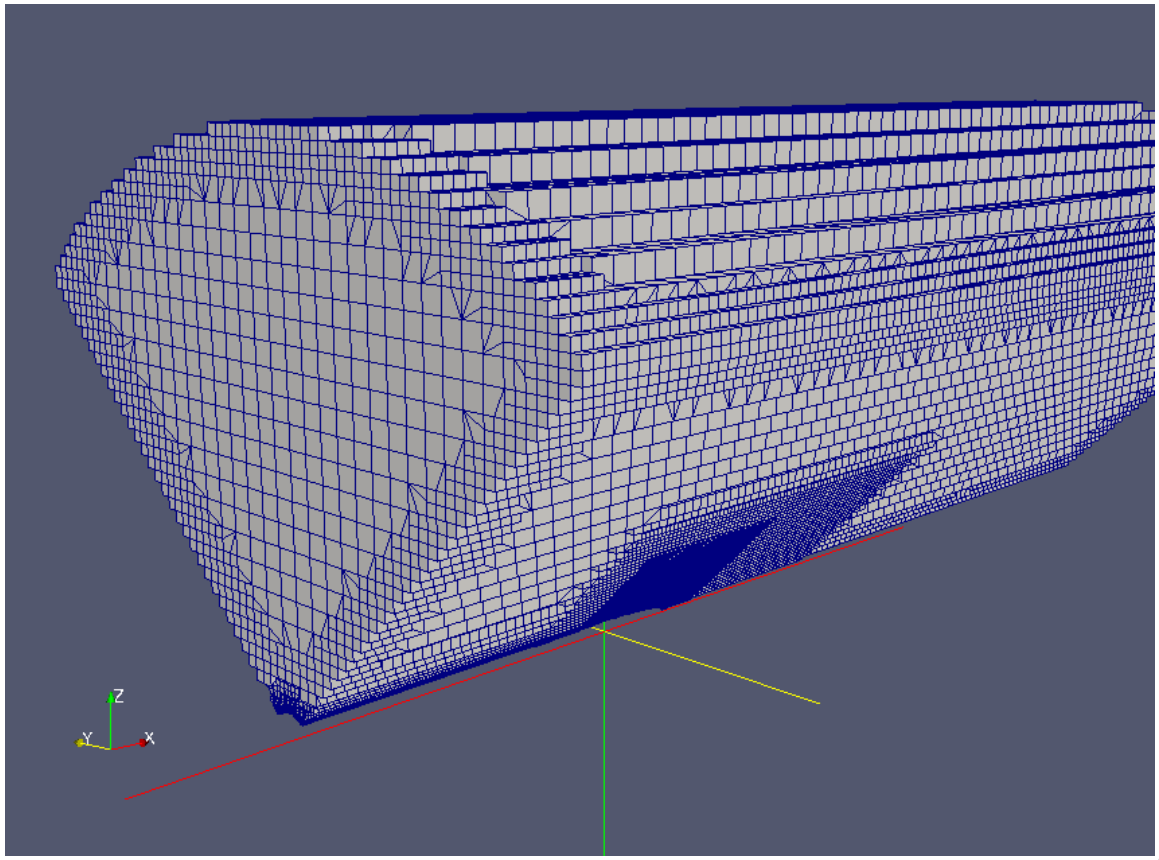
```
// Explicit feature edge refinement
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
    {
        file "Inlet.eMesh";
        level 1;
    }
    {
        file "Hub.eMesh";
        levels ((0.0 4) (0.003 4));
    }
    {
        file "Outlet.eMesh";
        level 1;
    }
    {
        file "Int.eMesh";
        level 1;
    }
    {
        file "Int2.eMesh";
        level 1;
    }
    {
        file "Ext.eMesh";
        level 1;
```

Each cells intersecting the edge is refined up to the selected level

Each cells intersecting the edge is refined up to the selected level, plus all the cells within a distance of 0.003 meters is refined up to the selected level

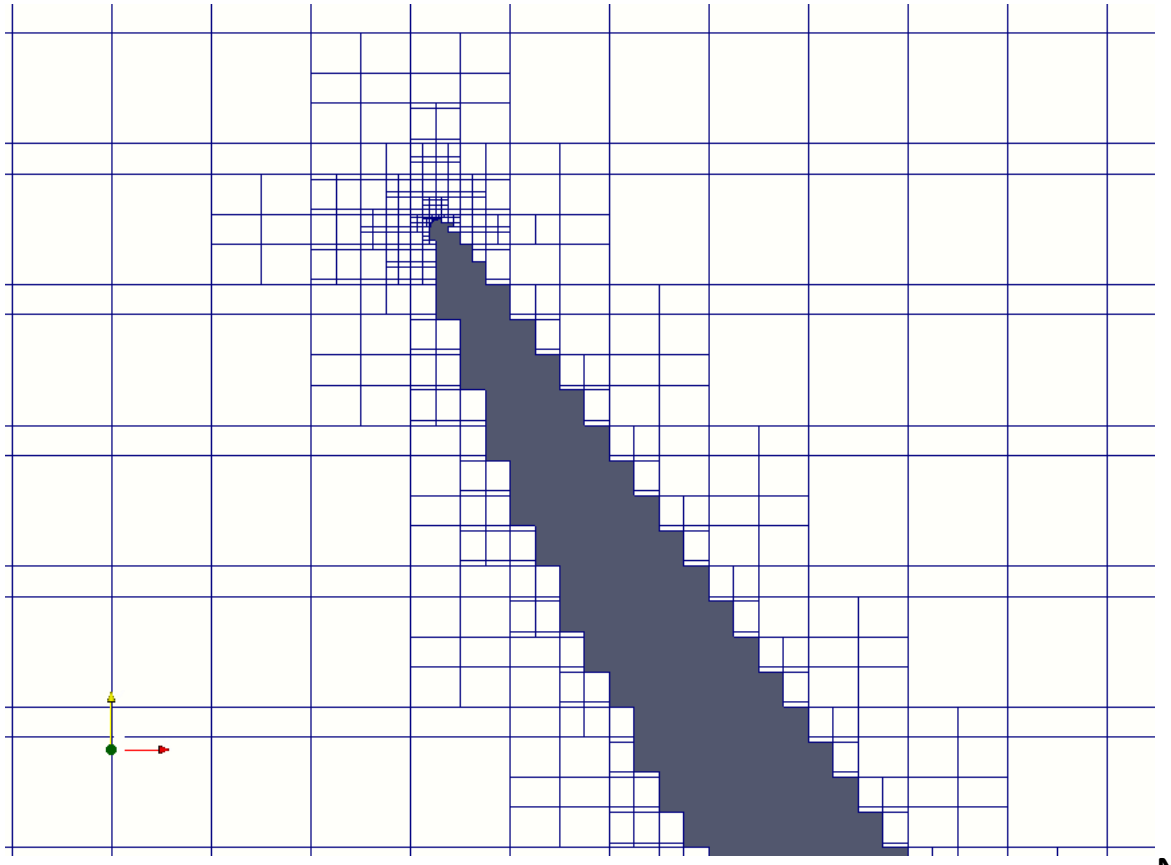
Hands-on session

- As usual with `snappyHexMesh`:
- Castellated domain, with selected refinements



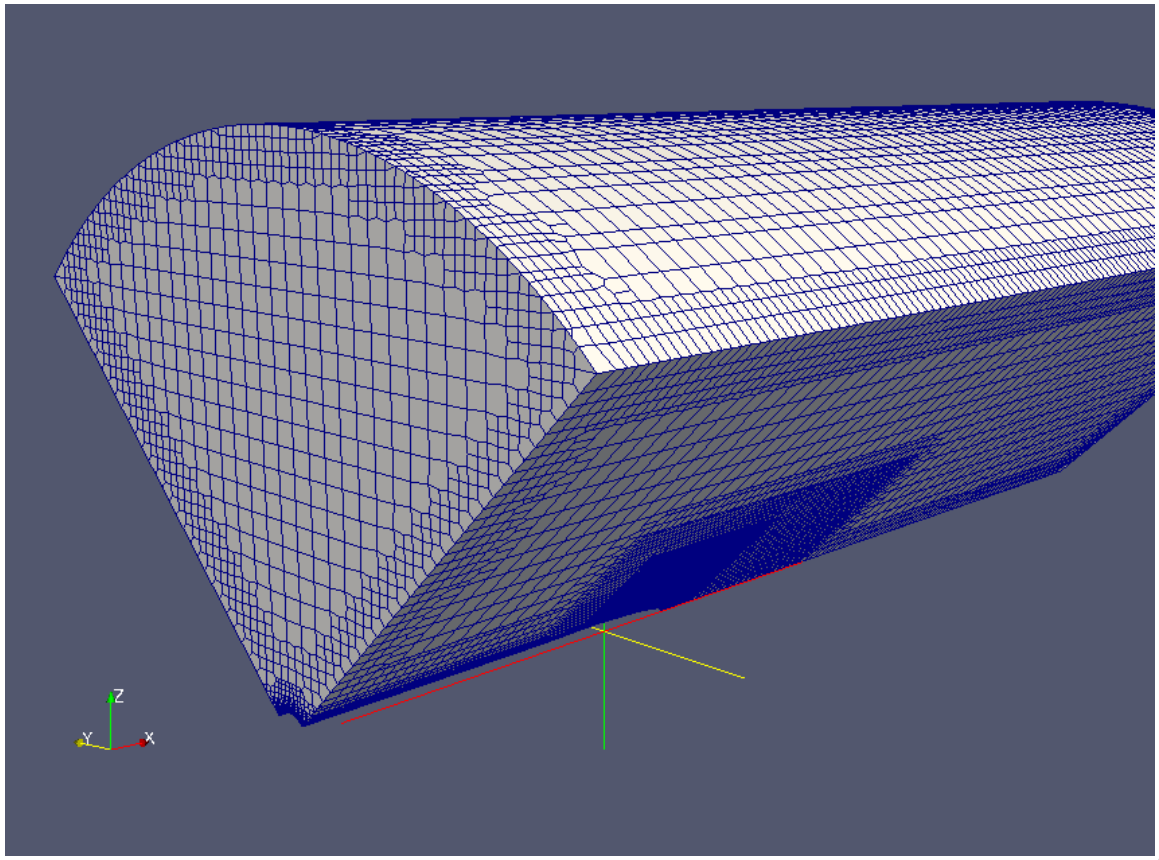
Hands-on session

- As usual with `snappyHexMesh`:
- Castellated domain, with selected refinements



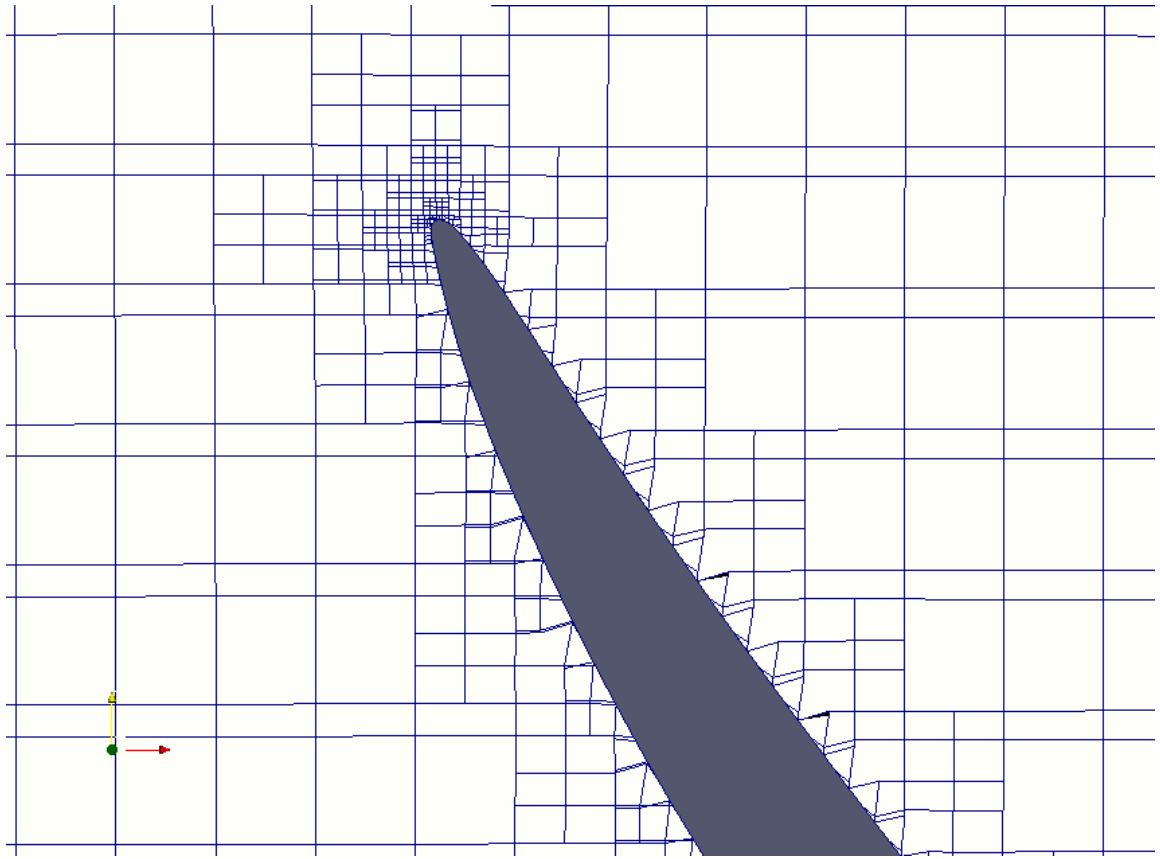
Hands-on session

- As usual with `snappyHexMesh`:
- Snapped domain, with selected refinements



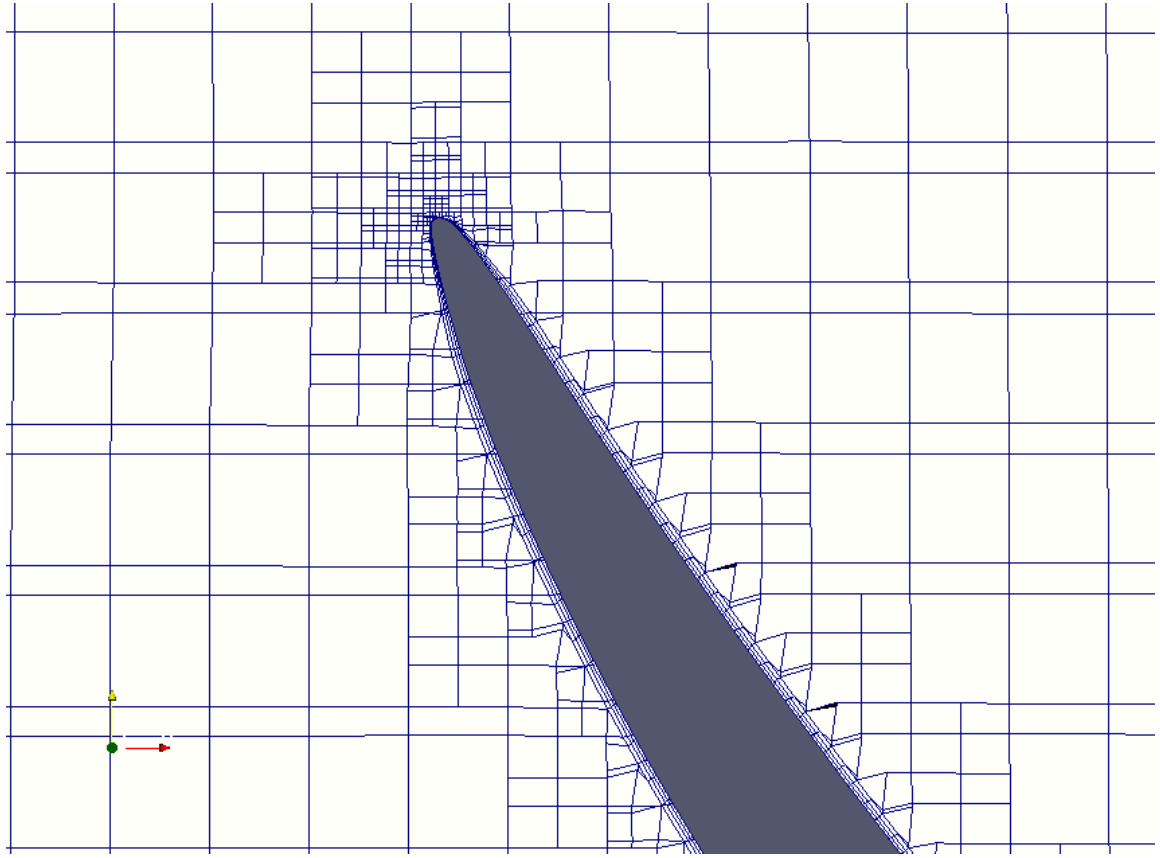
Hands-on session

- As usual with `snappyHexMesh`:
- Snapped domain, with selected refinements



Hands-on session

- As usual with `snappyHexMesh`:
- Prism layer Addition



Hands-on session

- Periodic Boundary Conditions by using the `createPatch` command:
  - `cyclic,` if conformal match (point – to – point equivalence) is granted for the interfaces (for instance with meshes imported from StarCCM+ or ad-hoc meshing tools)
  - `cyclicAMI,` when conformal matching is not possible (as with `snappyHexMesh`)
  - A custom `dict` file (`createPatchDict`) is necessary.

Hands-on session

- Periodic Boundary Conditions:

```
// Patches to create.
patches
(
    {
        // Name of new patch
        name Int_1_AMI;

        // Type of new patch
        patchInfo
        {
            type cyclicAMI;
        neighbourPatch Int_2_AMI;
            // Optional: explicitly set transformation tensor.
            // Used when matching and synchronising points.
            //transform translational;
            //separationVector (-2289 0 0);
            transform rotational;
            rotationAxis (1 0 0);
            rotationCentre (0 0 0);
        rotationAngle 90;
        }

        // How to construct: either from 'patches' or 'set'
        constructFrom patches;

        // If constructFrom = patches : names of patches. Wildcards allowed.
        patches (Int Int2);

    }
```

New name of the first periodic patch

and its neighbour (new name of the second patch)

Type of transformation, with optional axis and center

Original patches to be used for the definition of the periodic B.C. (both for the first periodic patch)

Hands-on session

- Periodic Boundary Conditions:

```
    // Name of new patch
    name Int_2_AMI;

    // Type of new patch
    patchInfo
    {
        type cyclicAMI;
    neighbourPatch Int_1_AMI;
        // Optional: explicitly set transformation tensor.
        // Used when matching and synchronising points.
        //transform translational;
        //separationVector (-2289 0 0);
        transform rotational;
        rotationAxis (1 0 0);
        rotationCentre (0 0 0);
        rotationAngle 90;
    }

    // How to construct: either from 'patches' or 'set'
    constructFrom patches;

    // If constructFrom = patches : names of patches. Wildcards allowed.
    patches (Int2);
}
```

New name of the second periodic patch

and its neighbour (new name of the first patch)

Type of transformation, with optional axis and center

Original patches to be used for the definition of the periodic B.C. (only the second original patch)

Hands-on session

- When you run the solution (OF3.0.1), check the weight of the Periodic Boundary Conditions:

```
AMI: Creating addressing and weights between 20 source faces and 40 target faces
AMI: Patch source sum(weights) min/max/average = 1, 1, 1
AMI: Patch target sum(weights) min/max/average = 1, 1, 1
```

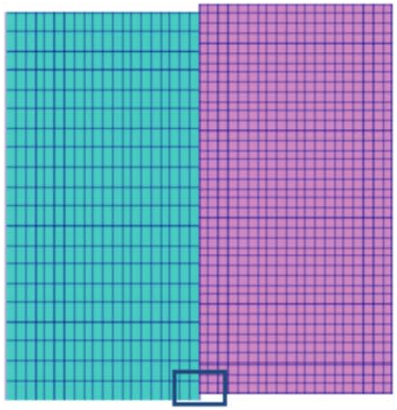
All weights equal to 1 is the ideal condition (patches conform perfectly)

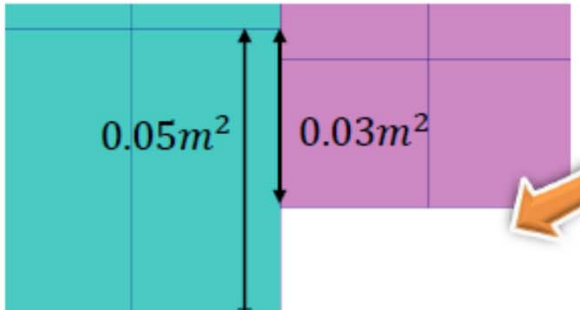
source is the first patch appearing in the `constant/polymesh/boundary` file
target is the second patch appearing in the `constant/polymesh/boundary` file

Hands-on session

- When you run the solution (OF3.0.1), check the weight of the Periodic Boundary Conditions:

```
AMI: Creating addressing and weights between 20 source faces and 40 target faces
AMI: Patch source sum(weights) min/max/average = 0.6, 1, 0.98
AMI: Patch target sum(weights) min/max/average = 0.2, 1, 0.98
```





Fraction of
the intersecting areas

$$sum(weights) = \frac{0.03}{0.05} = 0.6$$

Hands-on session

- Periodic Boundary Conditions:
  - Be careful to have weights higher than 0 (in case, change the interface surface mesh)!
  - in case of sliding meshes, weight should be higher than 0 for any relative position of the source/target patches.
  - A mitigation of the problem can be achieved by specifying:

```
lowWeightCorrection 0.2;
```

  - When a weight falls below 0.2 a zeroGradient B.C. is applied to those faces

Hands-on session

- Set the MultipleReferenceFrame (MRF)
  - `simpleFOAM` works with the Multiple Reference Frame if in the `constant` directory an appropriate `MRFProperties` dict is present.
  - In the `MRFProperties` file it is possible to specify Multiple Reference Frames to be applied to the entire domain or to portion of it, previously selected by using the `topoSet` command

Hands-on session

- Select the region (in this case the entire domain) to which the MRF is applied using the `topoSet` command with its dictionary

```
actions
(
    // Load initial cellSet
    {
        name    c0;
        type    cellSet;
        action  new;
        source cylinderToCell;
        sourceInfo
        {
           p1       (-10 0 0); // start point on cylinder axis
           p2       (10 0 0);   // end point on cylinder axis
           radius   10.0;
        }
    }

    {
        // Select based on cellSet
        name  Propeller;
        action  new;
        type    cellZoneSet;
        source setToCellZone;
        sourceInfo
        {
            set c0;           // name of cellSet
        }
    }
);
```

Create a cellSet using a cylinder

Assign to a cellZone (name Propeller) the cells previously selected

Hands-on session

- Set the MultipleReferenceFrame (MRF) with `MRFProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      MRFProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

MRF1
{
    cellZone    Propeller;
    active      yes;

    // Fixed patches (by default they 'move' with the MRF zone)
    nonRotatingPatches (Inlet Outlet Ext Int_1_AMI Int_2_AMI);
        origin      (0 0 0);
        axis        (1 0 0);
        omega       -157.0796327;
}

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

MRF is employed for the cellZone Propeller

Rotation Rate in radians per second

Hands-on session

- Set the MultipleReferenceFrame (MRF) with `MRFProperties`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      MRFProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

MRF1
{
    cellZone    Propeller;
    active      yes;

    // Fixed patches (by default they 'move' with the MRF zone)
    nonRotatingPatches (Inlet Outlet Ext Int_1_AMI Int_2_AMI);
        origin      (0 0 0);
        axis        (1 0 0);
        omega       -157.0796327;
}

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

```

BE CAREFUL! Inlet, Outlet and AMI patches must be `NonRotatingPatches`

Hands-on session

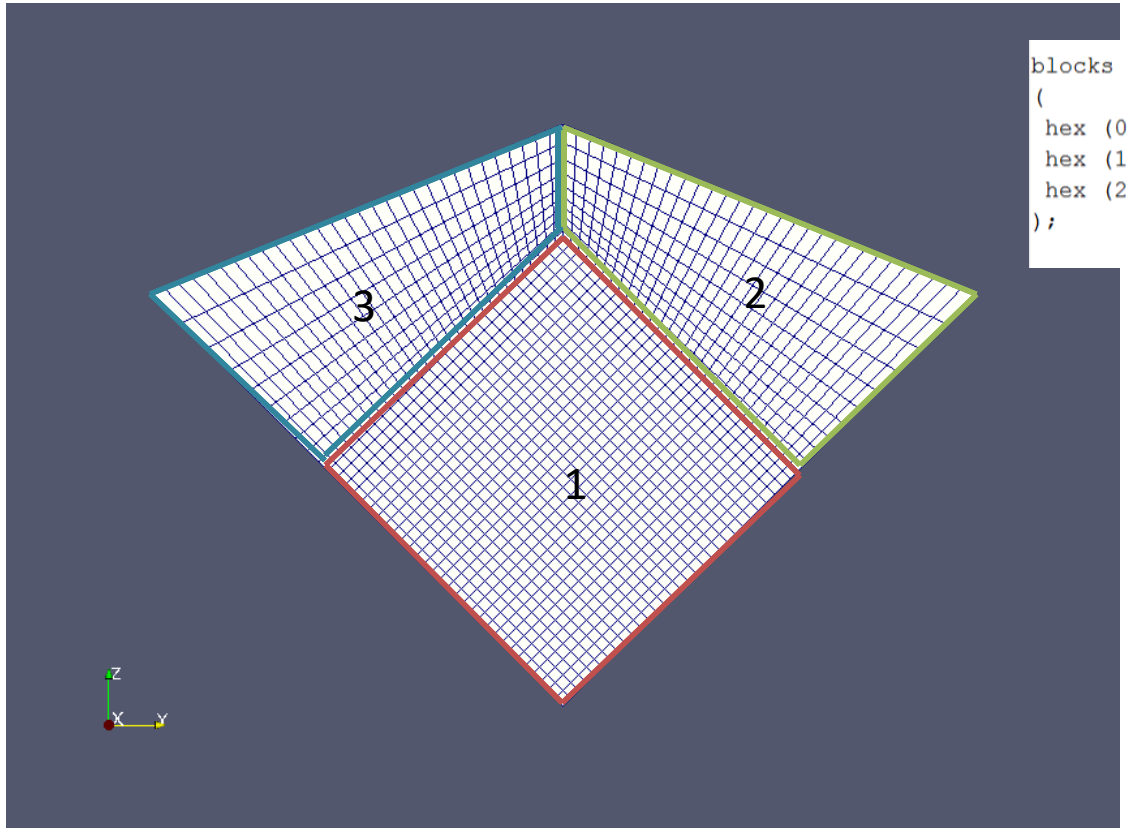
- Domain definition with `snappyHexMesh` + `blockMesh`
  - To improve the quality of the interfaces, by using `blockMesh` it is possible to directly define the “angular” computational domain, avoiding the snapping of cells in the interfaces (that could locally deform even planar patches and introduce some discrepancies in `cyclicAMI` definition)

Hands-on session

- Domain definition with `snappyHexMesh` + `blockMesh`
  - This is the best choice when the number of blade is equal to 4 (Angular sector of 90°, purely cubic cells)
  - In case of 3 or 5-6 blades, non cubic cells can be accepted on the light of the improvements on the interfaces

Hands-on session

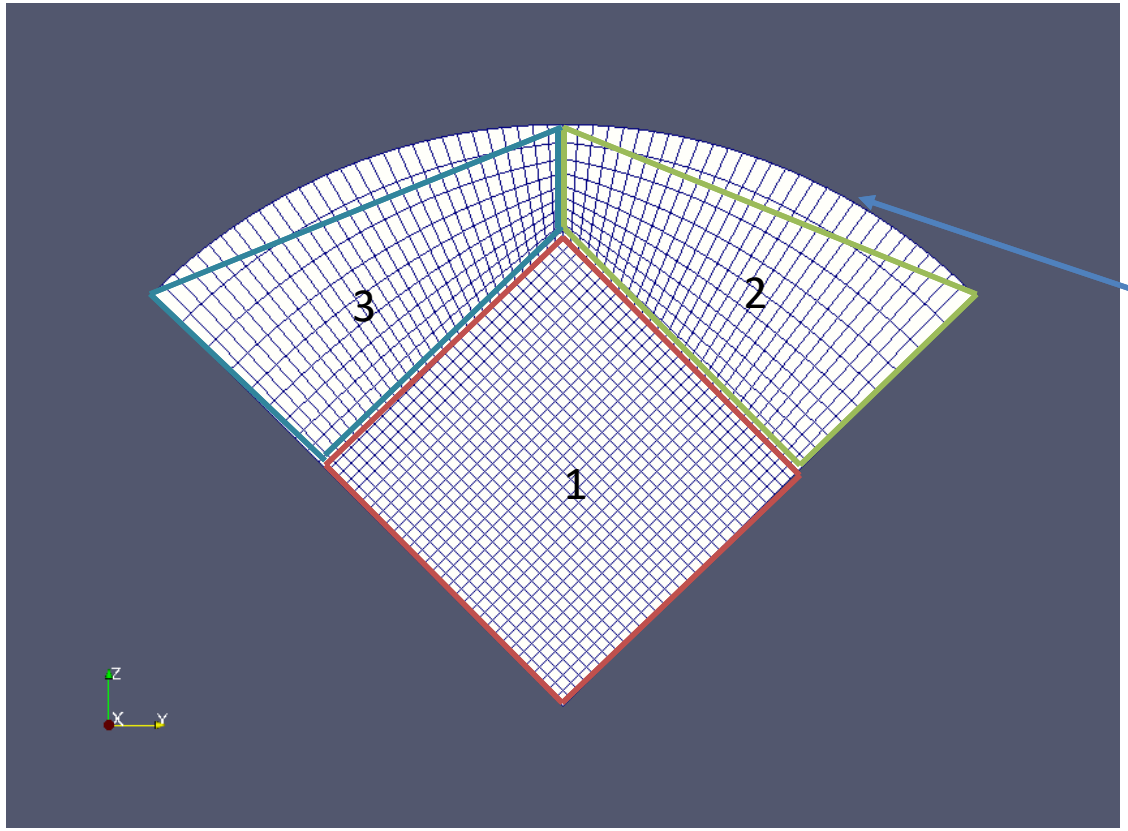
- Domain definition with `snappyHexMesh` + `blockMesh`
  - Initial blockMesh with 3 blocks




```
blocks
(
 hex (0 1 2 3 7 8 9 10)    (30 30 60) simpleGrading (1 1 1)
 hex (1 4 5 2 8 11 12 9)   (10 30 60) simpleGrading (3 1 1)
 hex (2 5 6 3 9 12 13 10) (10 30 60) simpleGrading (3 1 1)
);
```

Hands-on session

- Domain definition with `snappyHexMesh` + `blockMesh`
  - Initial blockMesh with 3 blocks

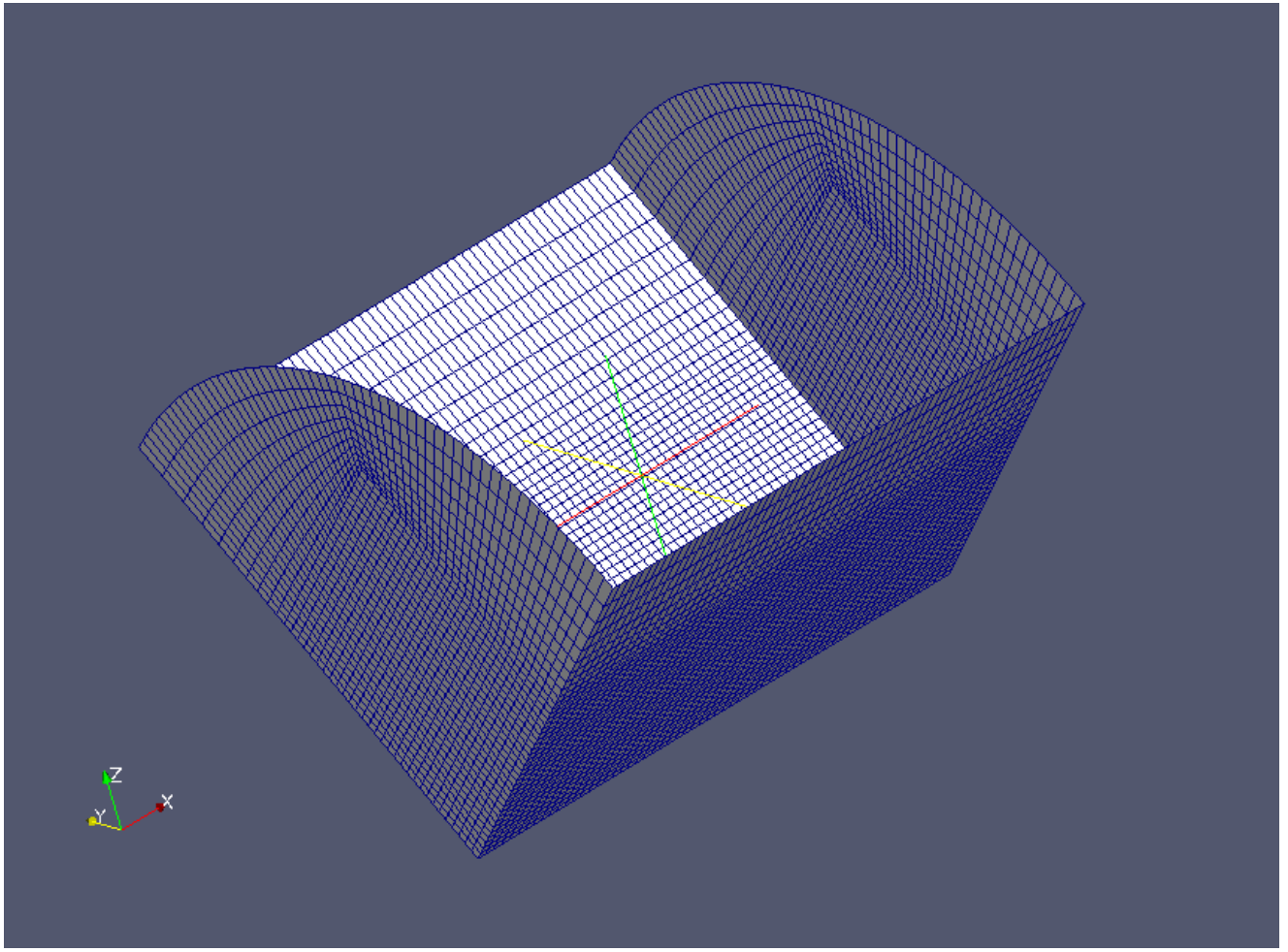


- Use the arc feature

```
edges
(

 arc 4 5 (-0.3  0.7 1.212435)
 arc 5 6 (-0.3  -0.7 1.212435)
 arc 11 12 (1.3 0.7 1.212435)
 arc 12 13 (1.3 -0.7 1.212435)



);
```

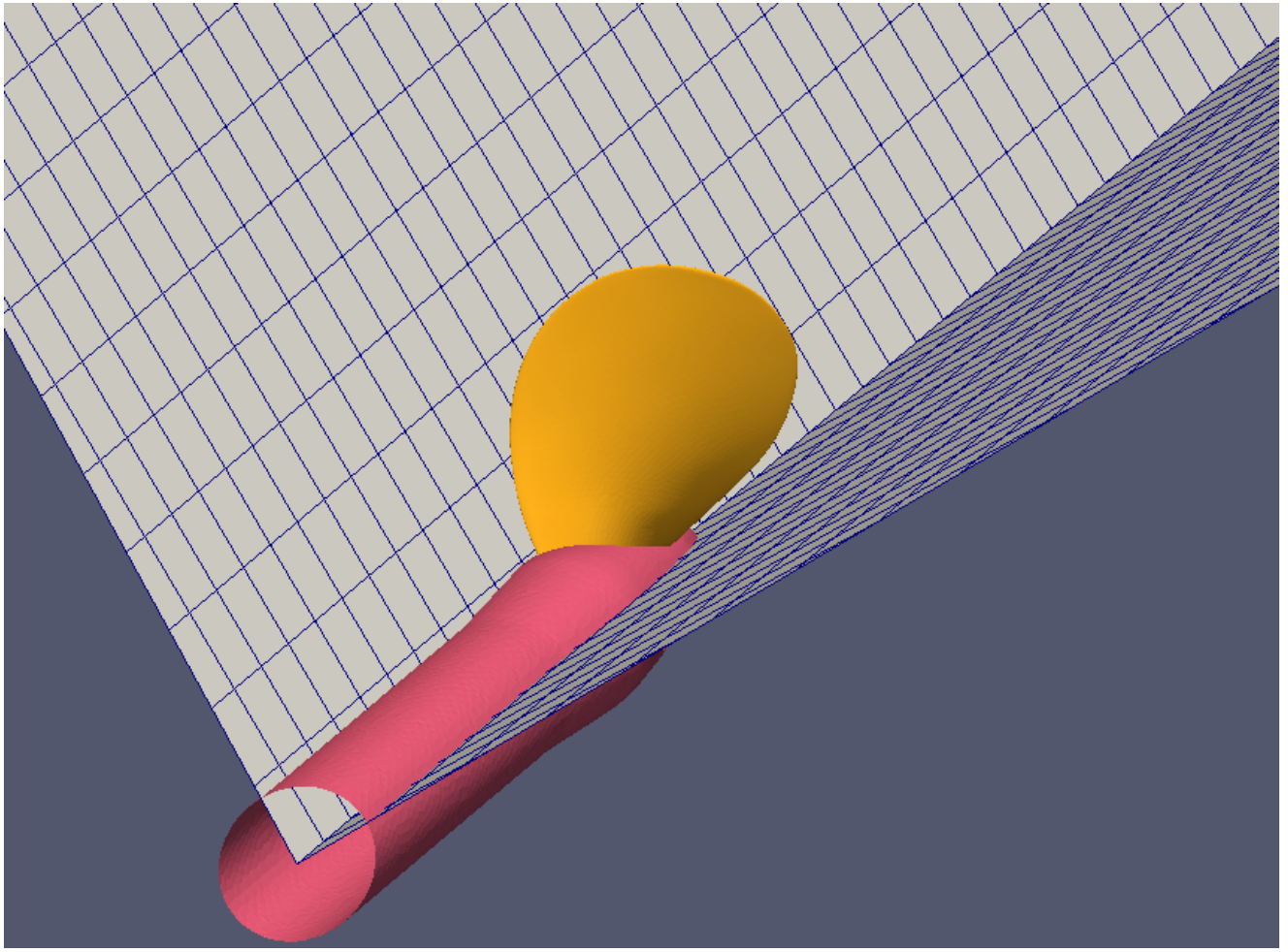
Hands-on session

- Domain definition with `snappyHexMesh` + `blockMesh`



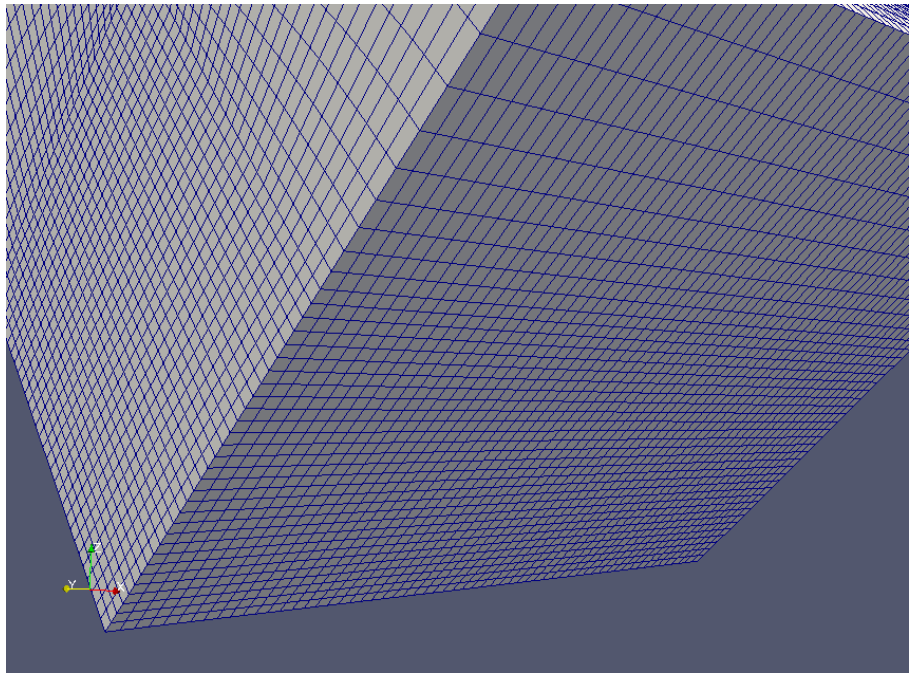
Hands-on session

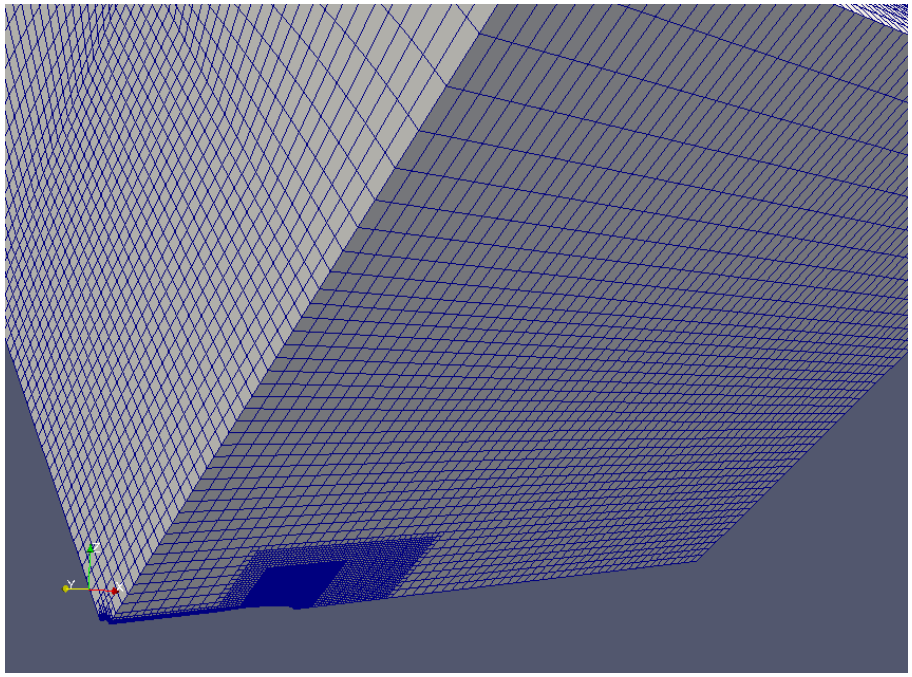
- Domain definition with `snappyHexMesh` + `blockMesh`



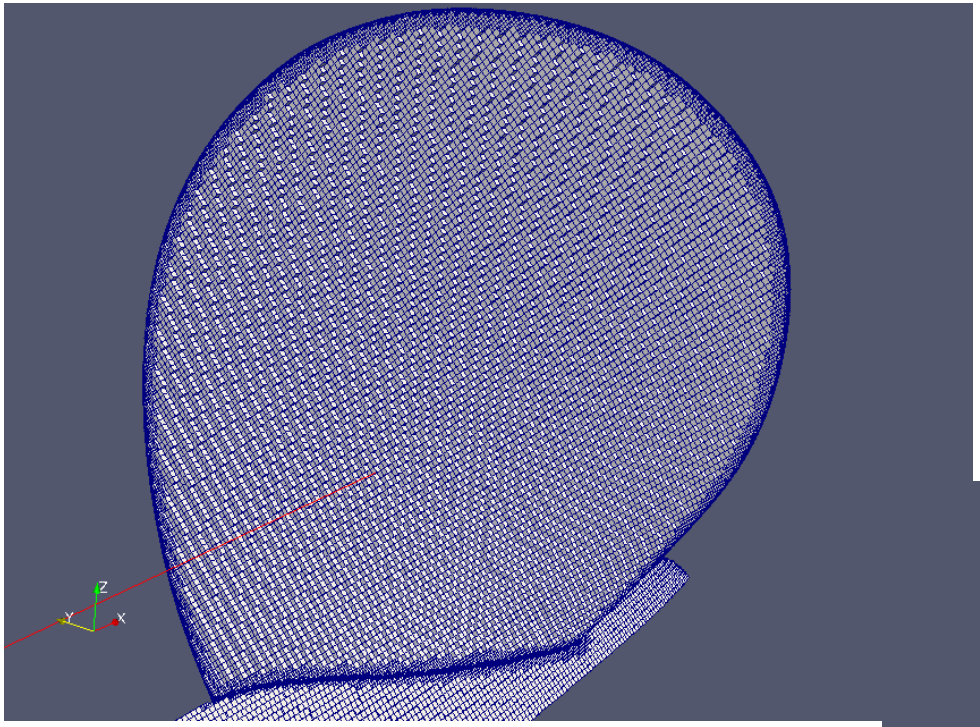
Hands-on session

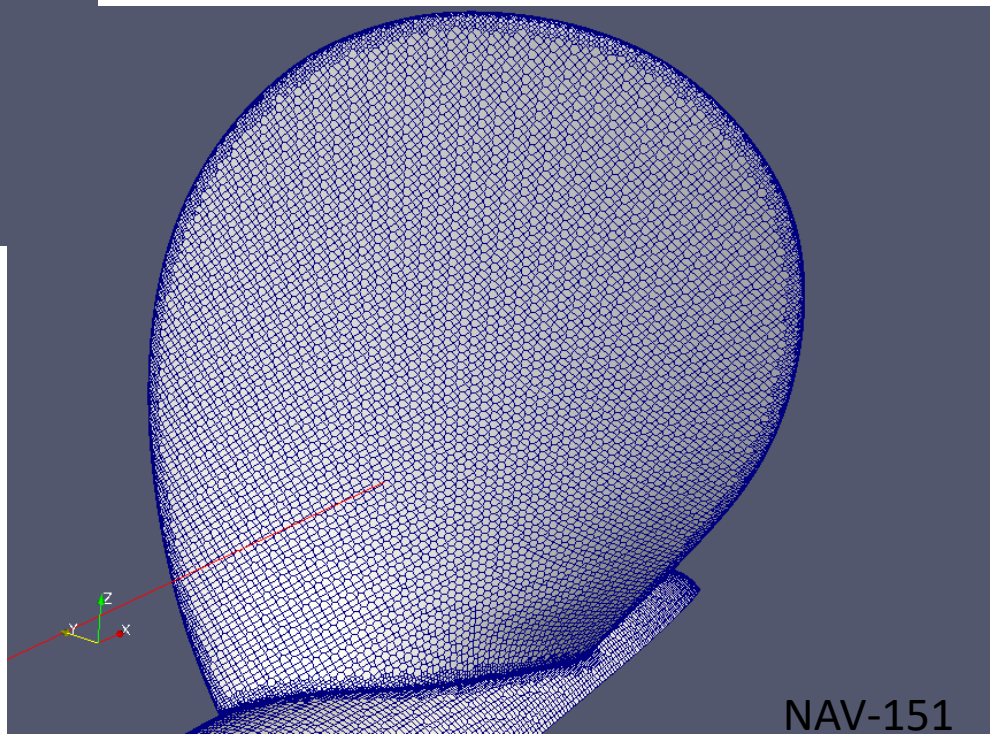
- Domain definition with `snappyHexMesh` + `blockMesh`





Hands-on session

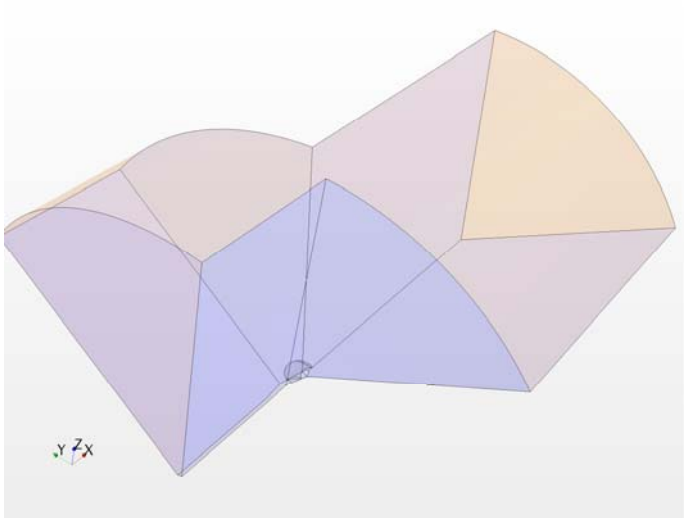


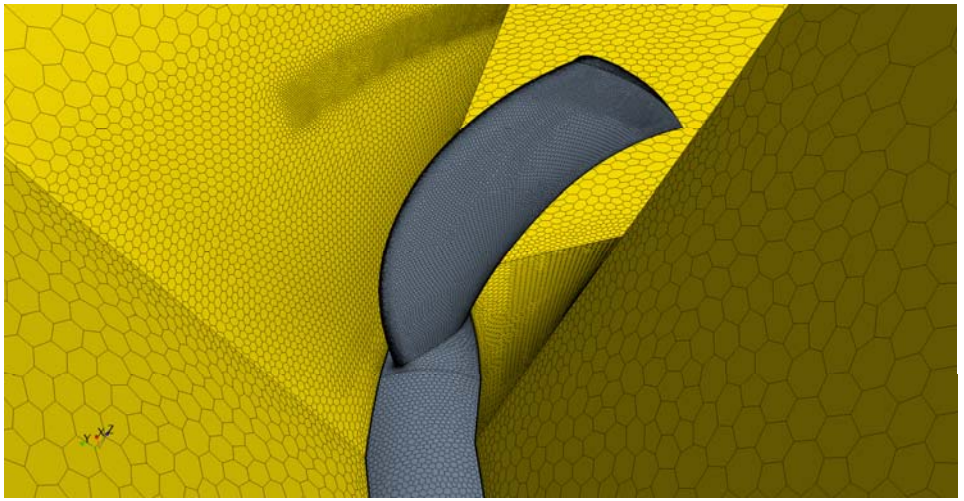


Hands-on session

- Why using `snappyHexMesh + blockMesh` approach?

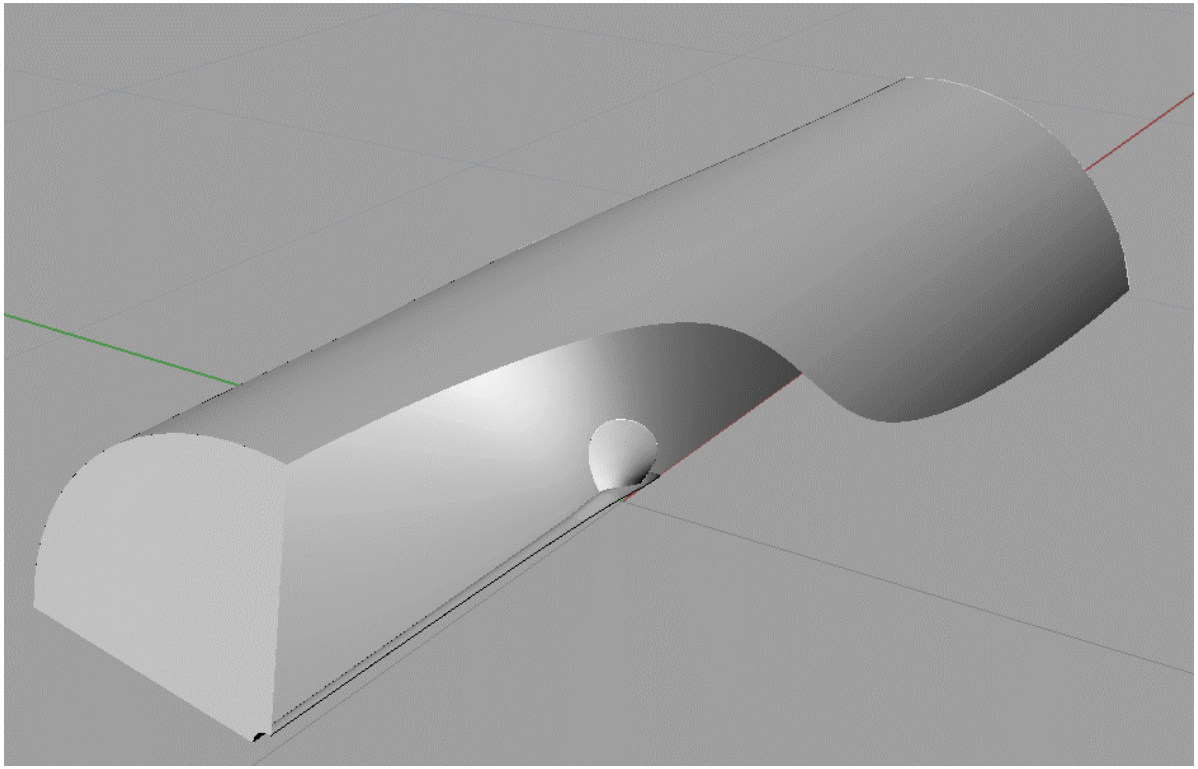
- When a propeller blade is highly skewed, it is useful to use the blade vane as the computational domain:





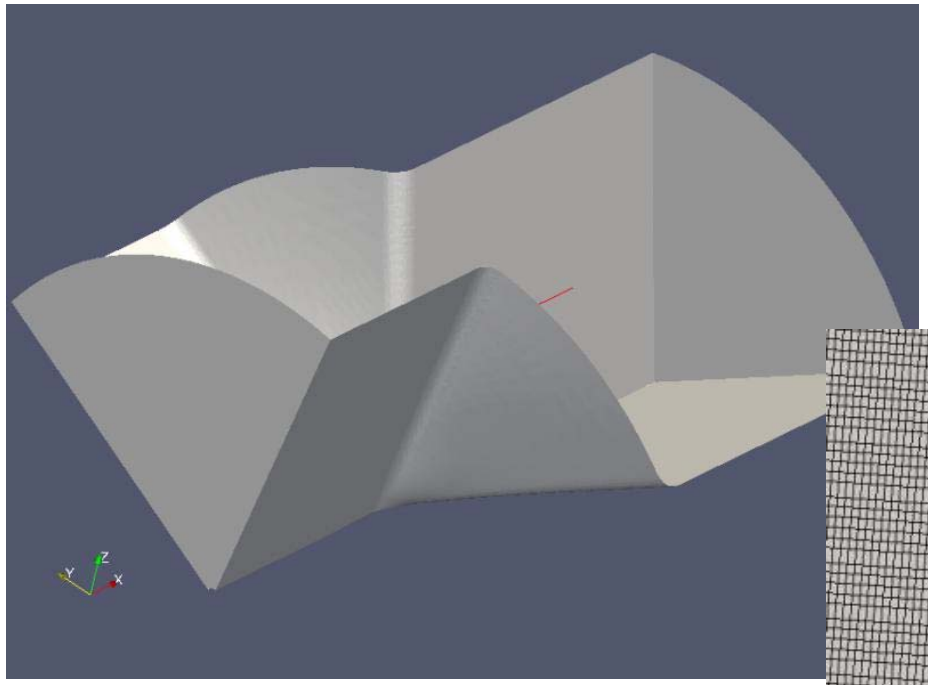
Hands-on session

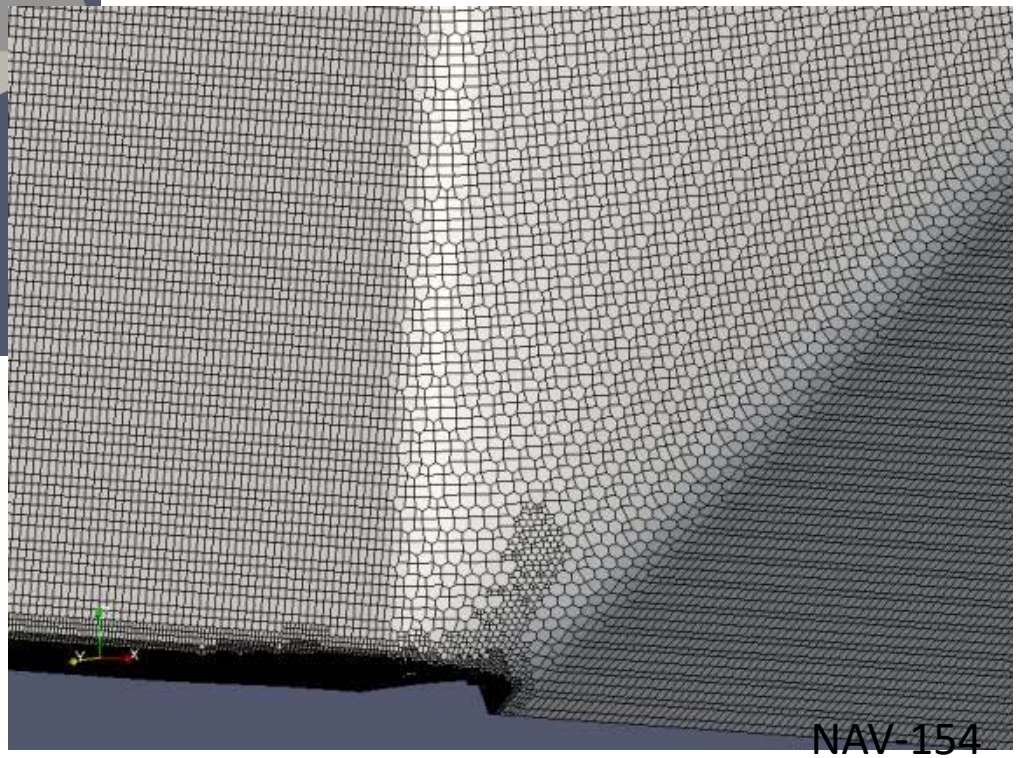
- The snappying process on this domain is problematic:



Hands-on session

- Apparently everything works well:





Hands-on session

- But typical errors when settings AMI interfaces are:
- 0 weight on AMI interfaces:

```
AMI: Creating addressing and weights between 11493 source faces and 11562 target faces
AMI: Patch source sum(weights) min/max/average = 0, 1.15886, 0.999861
AMI: Patch target sum(weights) min/max/average = 0, 1.48917, 0.999487
```

- Warning on tolerance matching, angle of rotations between the interfaces, even setting the right axis of rotation/angle, relaxing tolerance..

```
--> FOAM Warning :
    From function void Foam::cyclicAMIPolyPatch::calcTransforms(const primitivePatch&, const pointFie
    in file AMIInterpolation/patches/cyclicAMI/cyclicAMIPolyPatch/cyclicAMIPolyPatch.C at line 204
    Patch areas are not consistent within 10 % indicating a possible error in the specified angle of
    owner patch     : Interfaccia_AMI_1
    neighbour patch : Interfaccia_AMI_2
    angle           : -90 deg
    area error      : 31.3047 %    match tolerance : 0.1
```

Hands-on session

- But typical errors when settings AMI interfaces are:
- Non similar bounding boxes

```
Adding new patch Int_2_AMI as patch 10 from
{
    type            cyclicAMI;
    neighbourPatch  Int_1_AMI;
    transform       rotational;
    rotationAxis    ( 1 0 0 );
    rotationCentre  ( 0 0 0 );
}


Moving faces from patch Int_1 to patch 9
Moving faces from patch Int_2 to patch 10

Doing topology modification to order faces.

AMI: Creating addressing and weights between 50109 source faces and 52528 target faces
--> FOAM Warning :
    From function AMIMethod<SourcePatch, TargetPatch>::checkPatches()
    in file lnInclude/AMIMethod.C at line 57
    Source and target patch bounding boxes are not similar
    source box span     : (1.35892951488 0.602100236305 0.705581247807)
    target box span     : (1.35892951488 0.553761429124 0.717576680629)
    source box          : (-0.534949064255 -0.601774871349 -1.41759434052e-16) (0.8239
    target box          : (-0.534949064255 -0.136404545852 -1.1056390847e-16) (0.82398
    inflated target box : (-0.616623072351 -0.218078553948 -0.0816740080961) (0.905654


--> FOAM FATAL ERROR:
Unable to set source and target faces
```

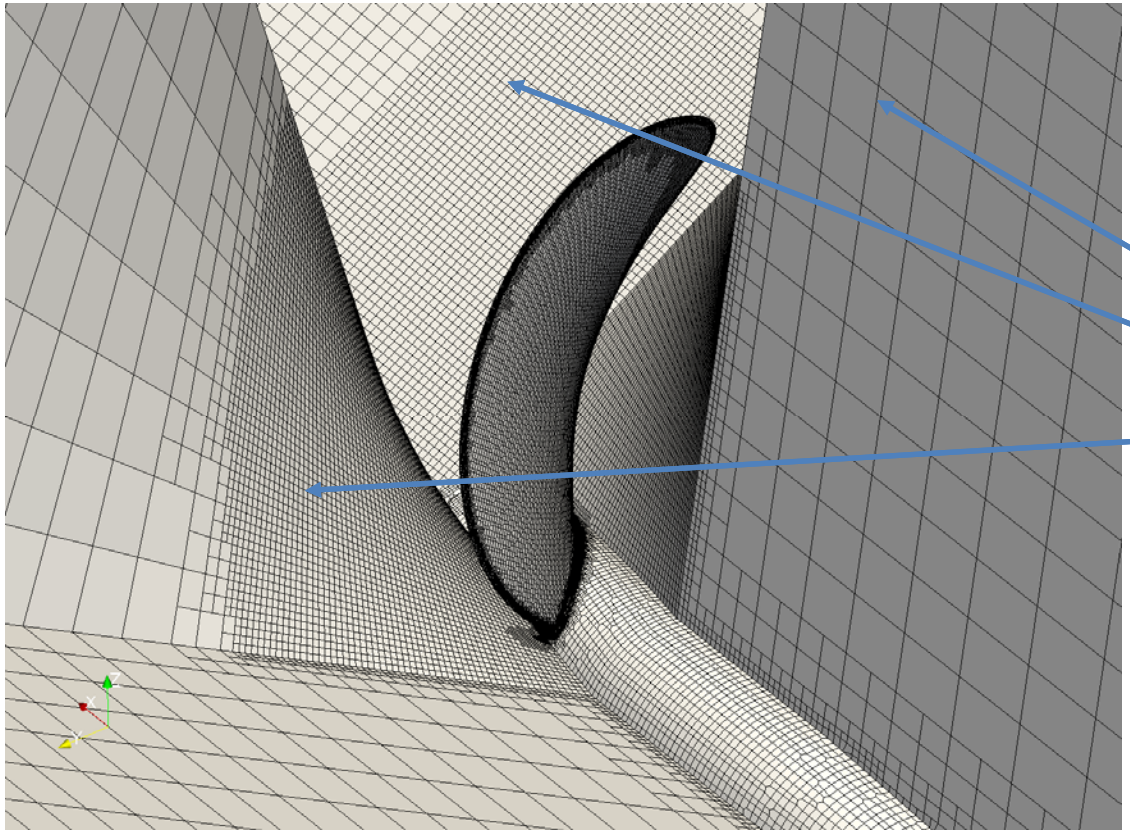
Hands-on session

- These errors are due to the deformations `snappyHexMesh` introduces on the curved interfaces (that, being curved, are snapped!), being AMI patches capable to account non planar interfaces (for instance when meshes are imported from other meshing tools),
- In our experience, sometimes, with very coarse meshes everything works!

- `snappyHexMesh + blockMesh` can alleviate these problems

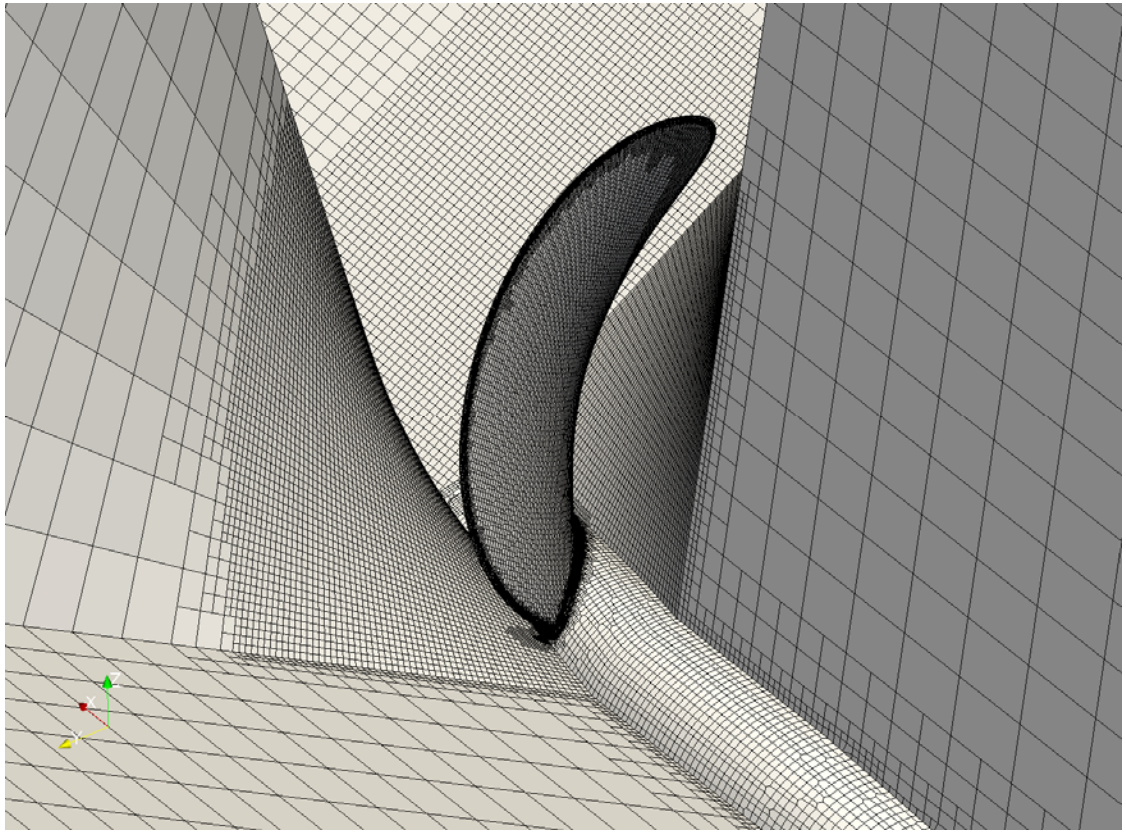
Hands-on session

- `snappyHexMesh` does not alter the original patches built with `blockMesh`, so the blade vane computational domain can be set up directly in `blockMesh`:



Hands-on session

- `snappyHexMesh` does not alter the original patches built with `blockMesh`, so the blade vane computational domain can be set up directly in `blockMesh`:

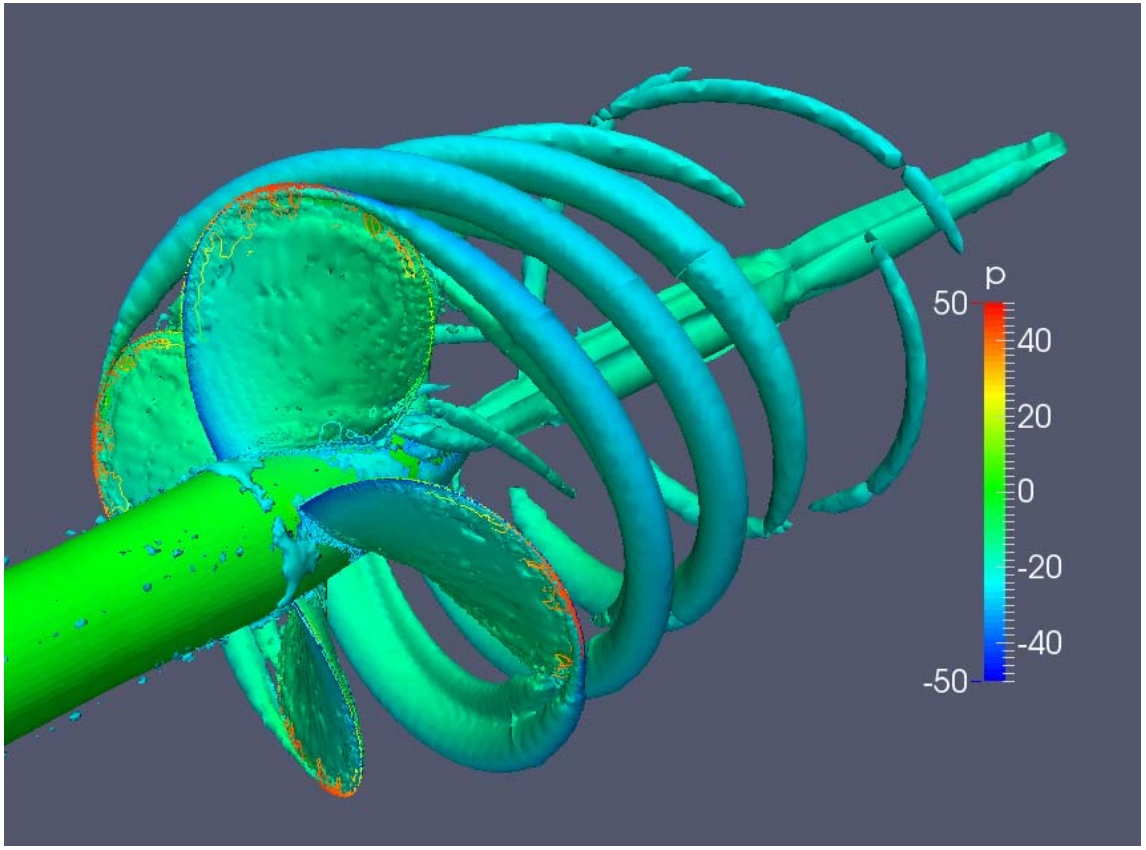


The only issue is that the base mesh is slighlty non-ortogonal and non-cubic (depending on the blade vane shape) for the castellation and the snappying process!

Hands-on session

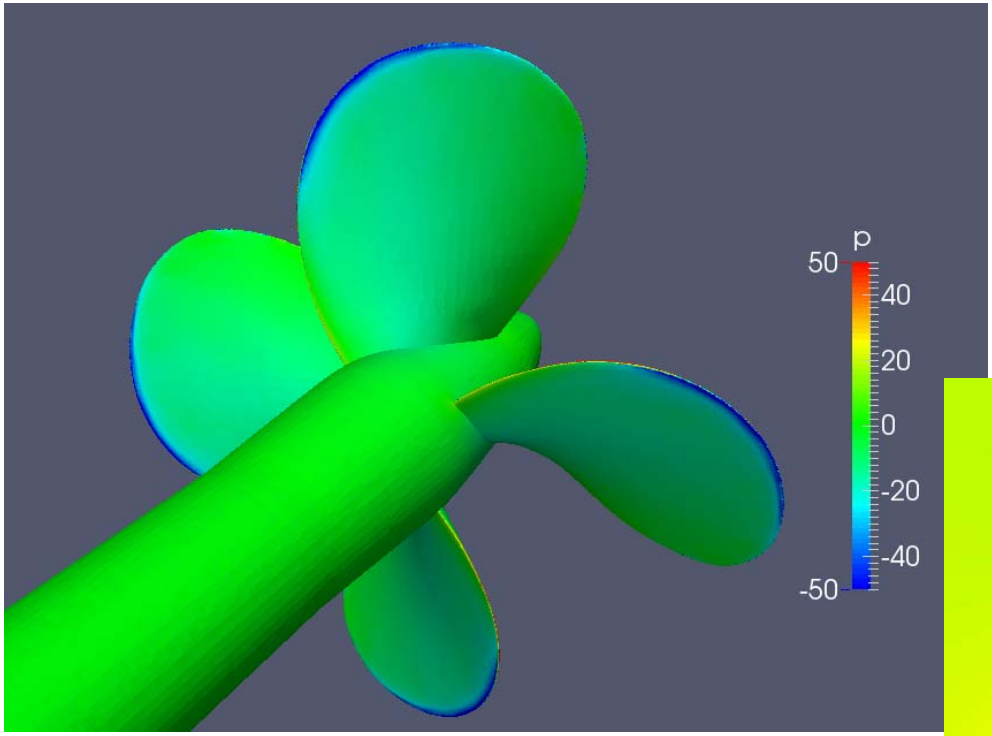
- Few results:

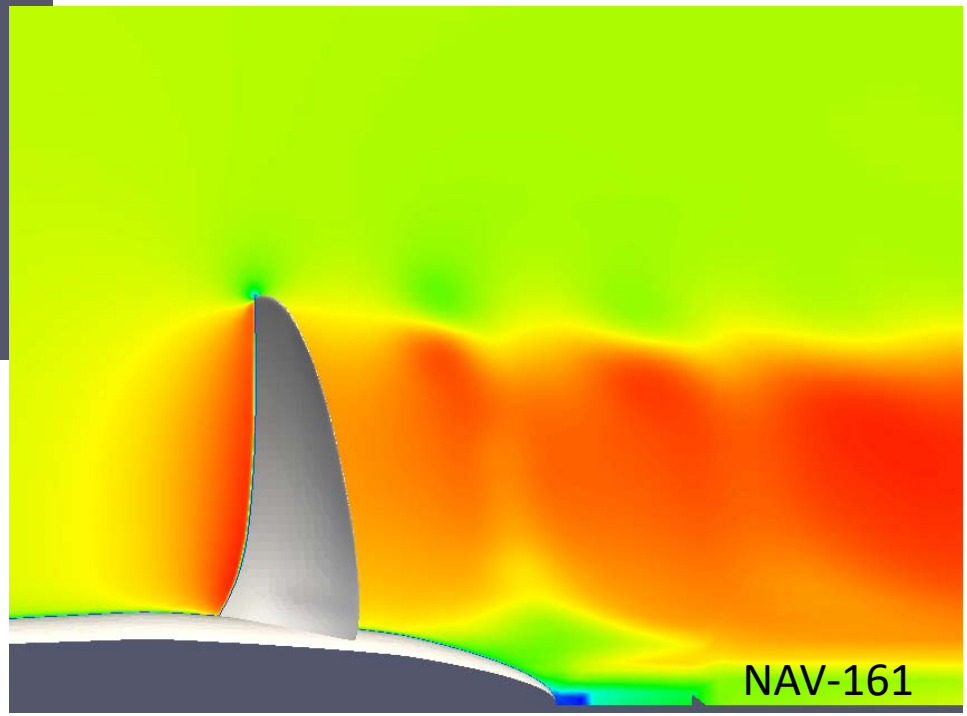
  Q-factor (command “Q” in the console) to trace vortexes



Hands-on session

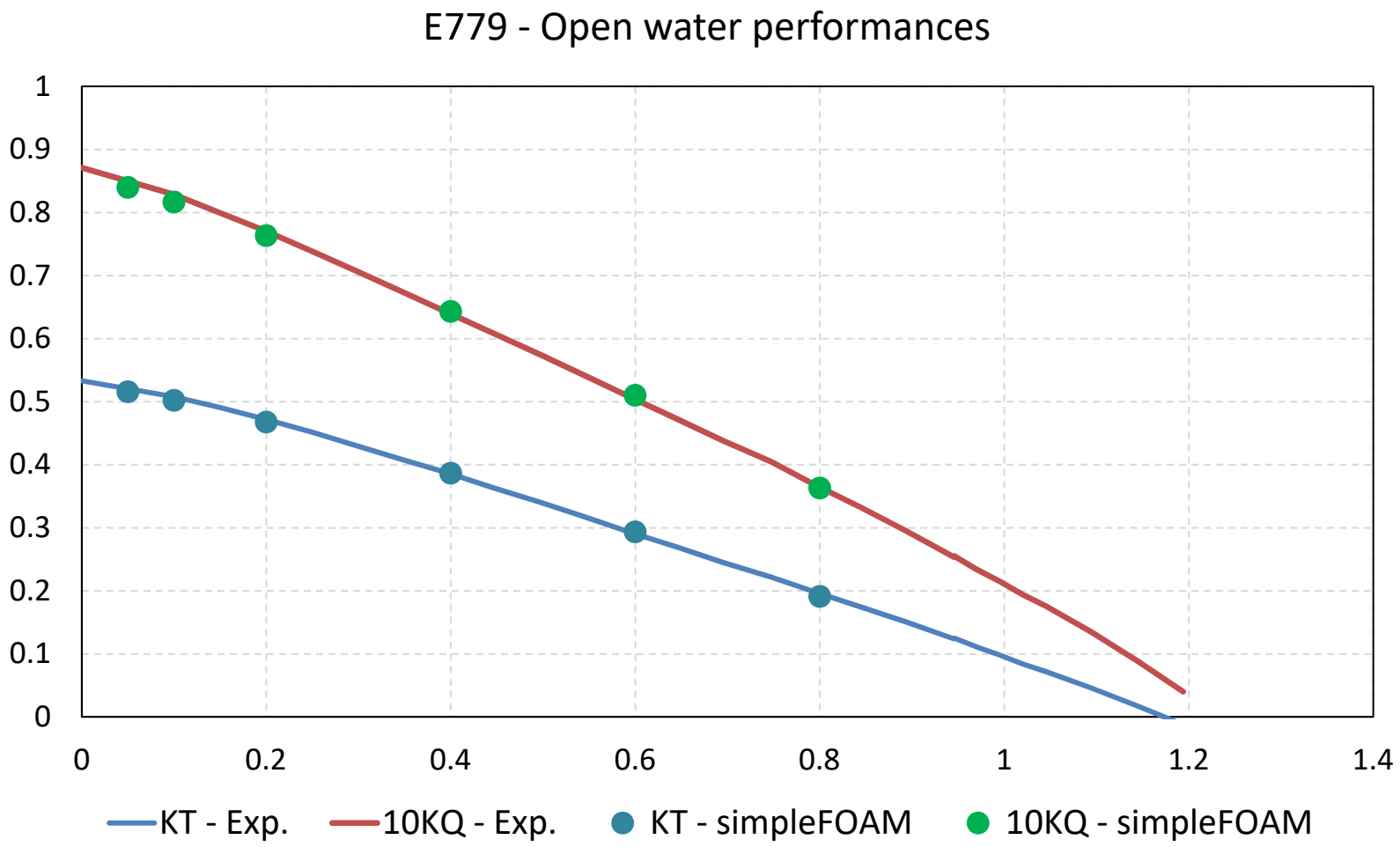
- Few results:





Hands-on session

- Few results:



Today's lecture

1. Introduction to the hydrodynamic naval problems
    1. Ship Resistance
    2. Propeller Performance
    3. Cavitation with OpenFOAM
2. Hands-on session.
    1. Drag resistance - KCS
    2. Open Water propeller performances - E779A
    3. Cavitation on a wing profile - NACA 0010

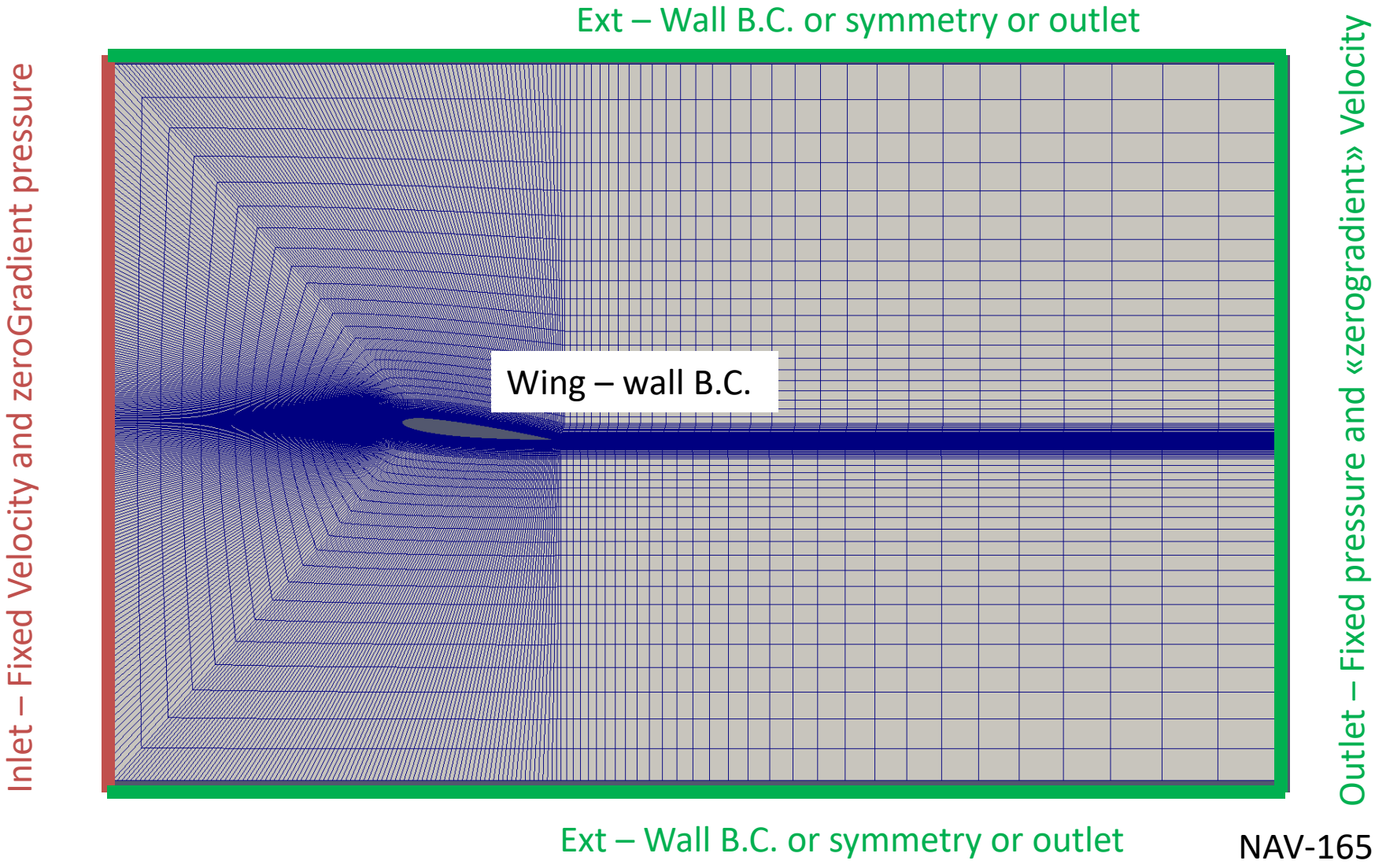
Hands-on session - Cavitation

- Define a Computational domain, fine enough to capture re-entrant jets, curvature…
- In this case: structured mesh around a NACA0010 wing profile, angle of attack = 4°
- `BlockMesh` to build the mesh (by using `spline` instead of strainght lines)

```
edges
(
   spline 5 4 (
        (0.125641 -0.158180 0.0)
        (0.128661 -0.158755 0.0)
        (0.131700 -0.159326 0.0)
        (0.134758 -0.159893 0.0)
        (0.137834 -0.160456 0.0)
        (0.140928 -0.161015 0.0)
```

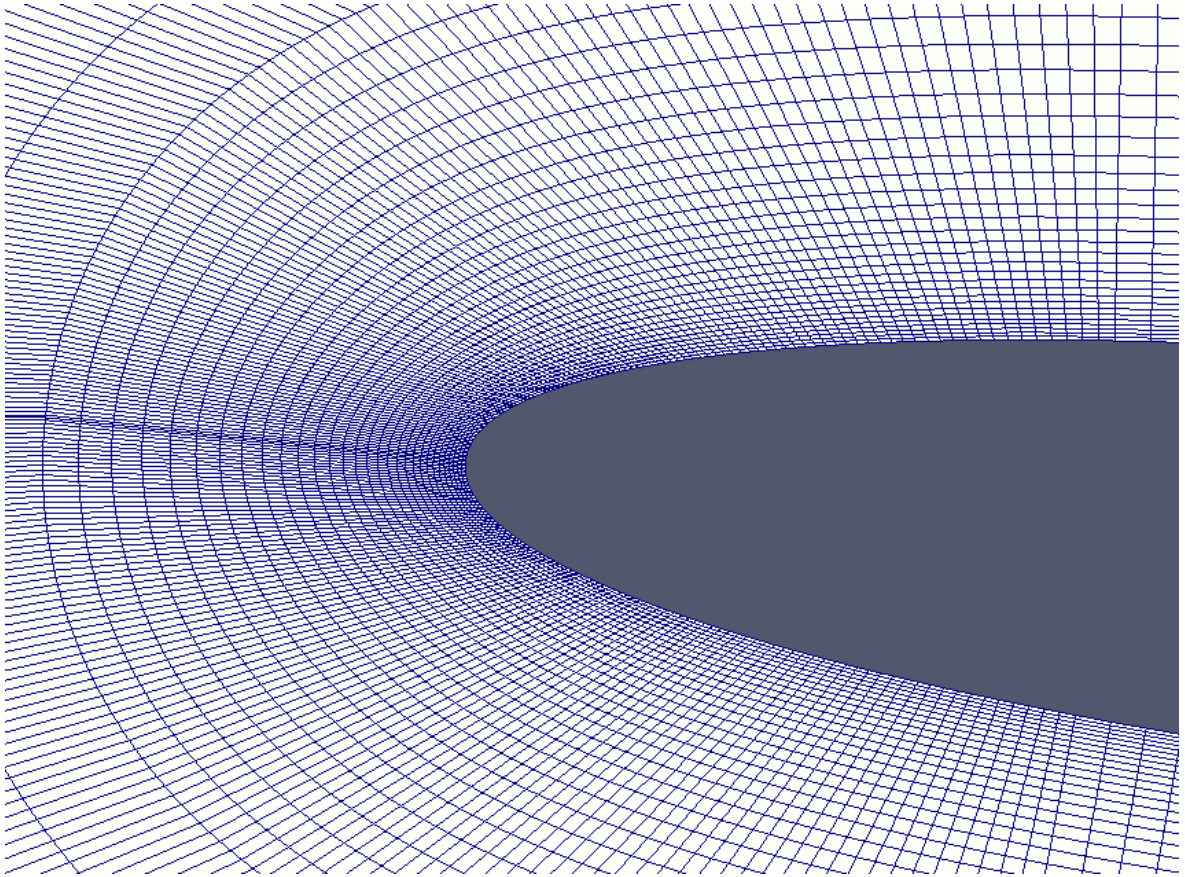
Hands-on session - Cavitation

- NACA0010



Hands-on session - Cavitation

- NACA0010 – Leading Edge details



Hands-on session - Cavitation

- Differences with respect to steady/single phase non cavitating calculations (`simpleFoam`, for instance):

- In the `0` folder:

  - `U`, `k`, `omega`, `epsilon`, `nut` file as for `simpleFoam`
  - `p_rgh` instead than `p`

Hands-on session - Cavitation

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

internalField   uniform 25000;

boundaryField
{
    Inlet
    {
        type            fixedFluxPressure;
    }

    Outlet
    {
        type            fixedValue;
        value           uniform 25000;
    }

    Ext
    {
        type            fixedValue;
        value           uniform 25000;
    }

    Wing
    {
        type            fixedFluxPressure;
    }
```

Pressure, not pressure/density!

Reference pressure, to be selected on the basis of the required cavitation Number (`internalField` and all the `Outlet` B.C.)

Hands-on session - Cavitation

- Differences with respect to steady/single phase non cavitating calculations (`simpleFoam`, for instance):

- In the `0` folder:
  - `U`, `k`, `omega`, `epsilon` file as for `simpleFoam`
  - `p_rgh` instead than `p`
  - An additional file (`alpha.water`) to set the initial conditions about phases (only water)

Hands-on session - Cavitation

```
dimensions      [0 0 0 0 0];

internalField   uniform 1;

boundaryField
{
    Inlet
    {
        type            fixedValue;
        value           uniform 1;
    }

    Outlet
    {
        type            fixedValue;
        value           uniform 1;
    }

    Wing
    {
        type            zeroGradient;
    }

    Ext
    {
    type            fixedValue;
        value           uniform 1;

    }
```

Only water!

Only water!

Only water!

Compare, for instance, commands necessary for free surface simulation, to set `alpha.water` on boundary conditions

Hands-on session - Cavitation

- Differences with respect to steady/single phase non cavitating calculations (`simpleFoam`, for instance):

- In the `constant` folder:

  - g file for the specification of the gravity acceleration (0 in this 2D case)

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 0 0);
```

Hands-on session - Cavitation

- Differences with respect to steady/single phase non cavitating calculations (`simpleFoam`, for instance):

- In the `constant` folder:

  - `transportPropieties` file with details of the phases

Hands-on session - Cavitation

```
16
17 phases (water vapour);
18
19 phaseChangeTwoPhaseMixture SchnerrSauer;
20
21 pSat             [1 -1 -2 0 0 0 0] 3200;   // Saturation pressure
22
23 sigma            [1 0 -2 0 0 0 0]  0; //0.07;
24
25 water
26 {
27     transportModel Newtonian;
28     nu              [0 2 -1 0 0 0 0] 1e-06;
29     rho             [1 -3 0 0 0 0 0] 1000;
30 }
31
32 vapour
33 {
34     transportModel Newtonian;
35     nu              [0 2 -1 0 0 0 0] 4.273e-04;
36     rho             [1 -3 0 0 0 0 0] 0.02308;
37 }
38
39 SchnerrSauerCoeffs
40 {
41     n               n       [0 -3 0 0 0 0 0]     1.6e+13;
42     dNuc            dNuc    [0 1 0 0 0 0 0]      2.0e-06;
43     Cc              Cc      [0 0 0 0 0 0 0]      1;
44     Cv              Cv      [0 0 0 0 0 0 0]      1;
45 }
46
47
48 // ************************************************************************* //
```

Name of the phases

Proprieties of the phases

n_0

$2R$

Hands-on session - Cavitation

- Differences with respect to steady/single phase non cavitating calculations (`simpleFoam`, for instance):

- In the `system` folder:

  - transient solver, so specification of the time discretization in the `fvScheme` file

  - numerical schemes for the fraction `alpha`

  - numerical solvers for `alpha`

Hands-on session - Cavitation

```
ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(U)         cellLimited Gauss linear 1;
}

divSchemes
{
    default             none;

    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,k)          Gauss upwind;
    div(phi,epsilon)    Gauss upwind;
    div(phi,omega)    Gauss upwind;

    div((muEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear limited corrected 0.33;
}
```

Temporal discretization

Numerical Schemes for the transport equation for `alpha`

Hands-on session - Cavitation

```
solvers
{
    "alpha.water.*"
    {
        cAlpha          0;
        nAlphaCorr      2;
        nAlphaSubCycles 1;

        MULESCorr       yes;
        nLimiterIter    5;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
        maxIter         10;
        minIter         1;
    };
```

Parameters of the interFOAM class, to control sharpness of the interface and number of correction cycles

```
PIMPLE
{
    correctPhi          yes;
    nOuterCcorrectors   1;
    nCorrectors         1;
    nNonOrthogonalCorrectors 1;
}
```

PIMPLE-based transient solver

Hands-on session - Cavitation

```
application     interPhaseChangeFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         0.5;

deltaT          1e-6;
writeControl    adjustableRunTime;
writeInterval   0.005;
adjustTimeStep  yes;
maxCo           0.75;
maxAlphaCo      0.5;
```

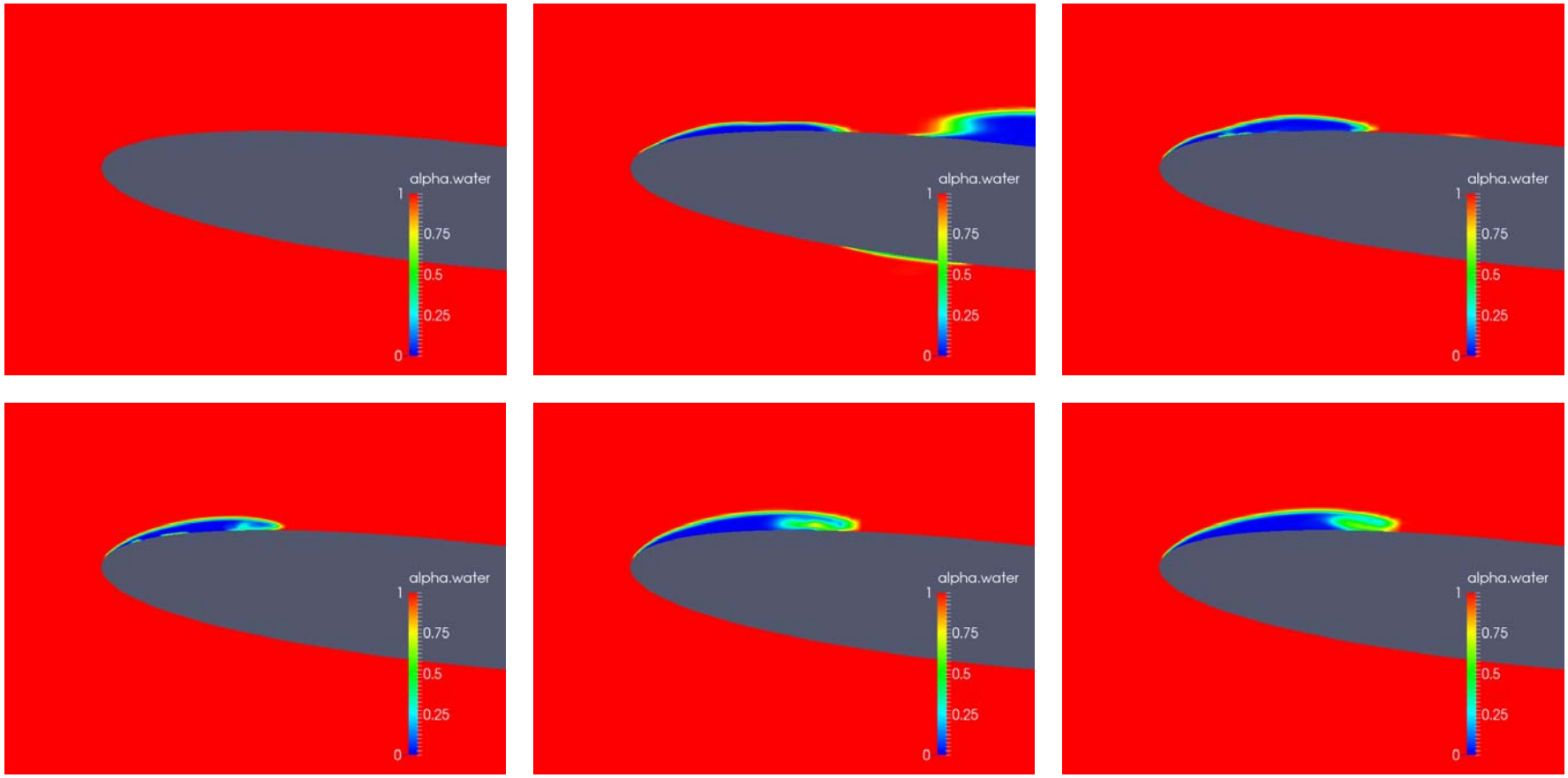
BE CAREFUL!

Cavitation is a transient process even if steady inflow conditions are considered.

A sufficiently low Courant number should be selected

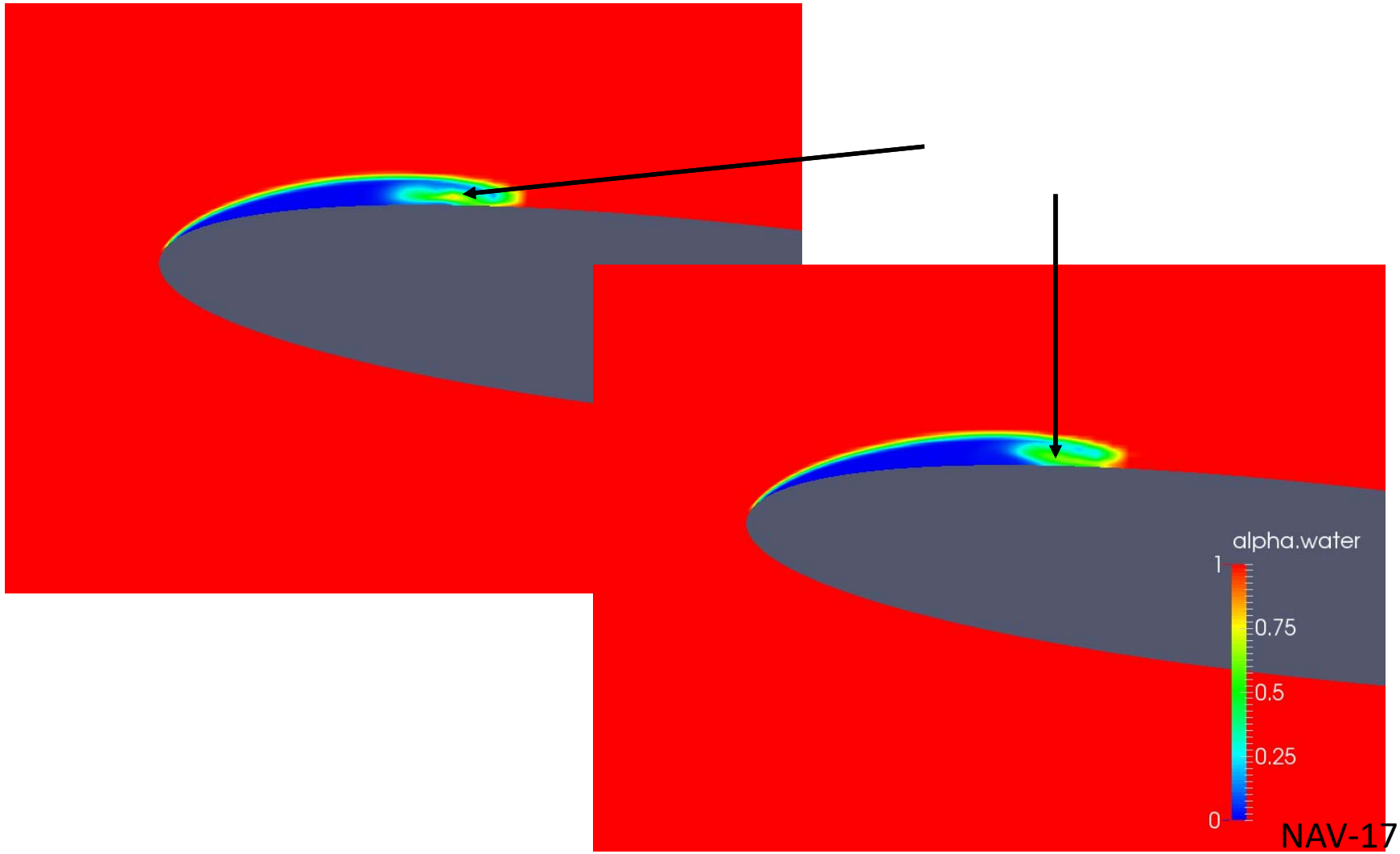
Hands-on session - Cavitation

- Some results:



Hands-on session - Cavitation

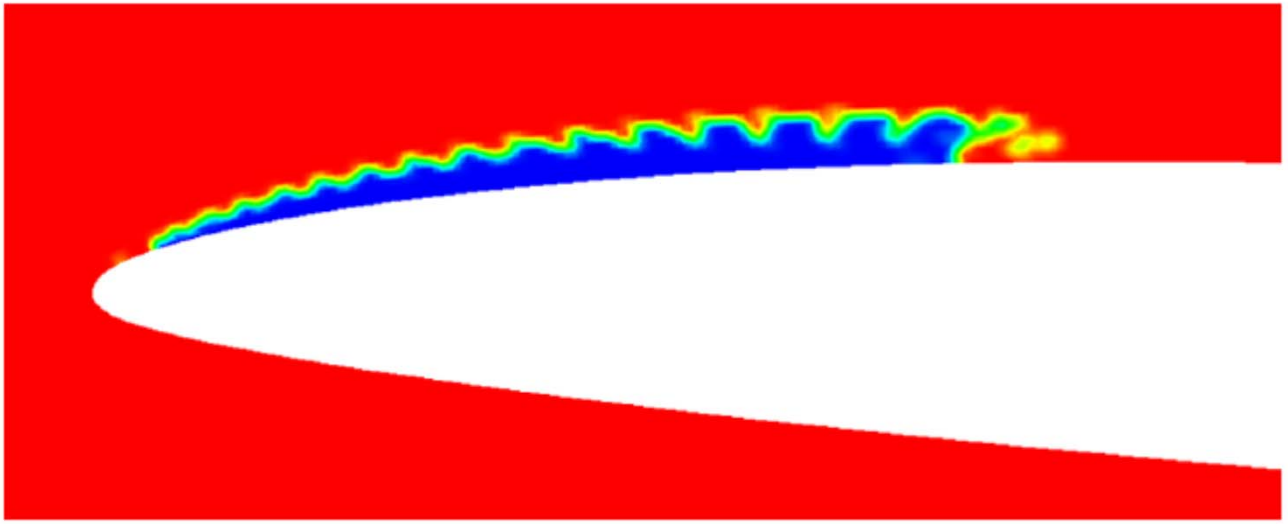
- Some results:



Hands-on session - Cavitation

- Some results:

BE CAREFUL! Excessive artificial sharpening (`cAlpha > 1`) with a non-adequate mesh



Hands-on session

- Training material:

Naval/KCS

Naval/E779A

Naval/Cav_2D

Thank you for your attention