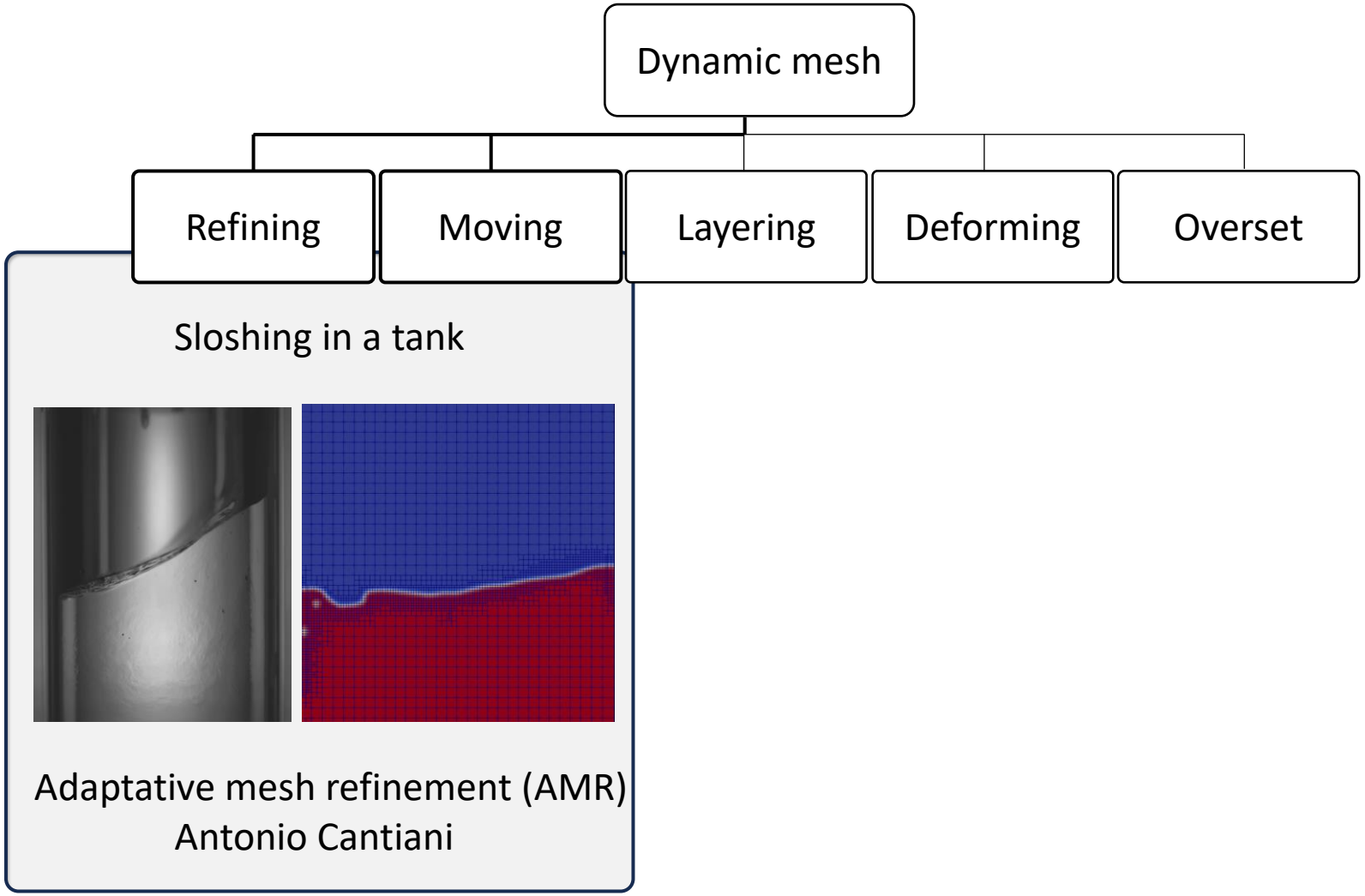
The von Karman Institute for Fluid Dynamics

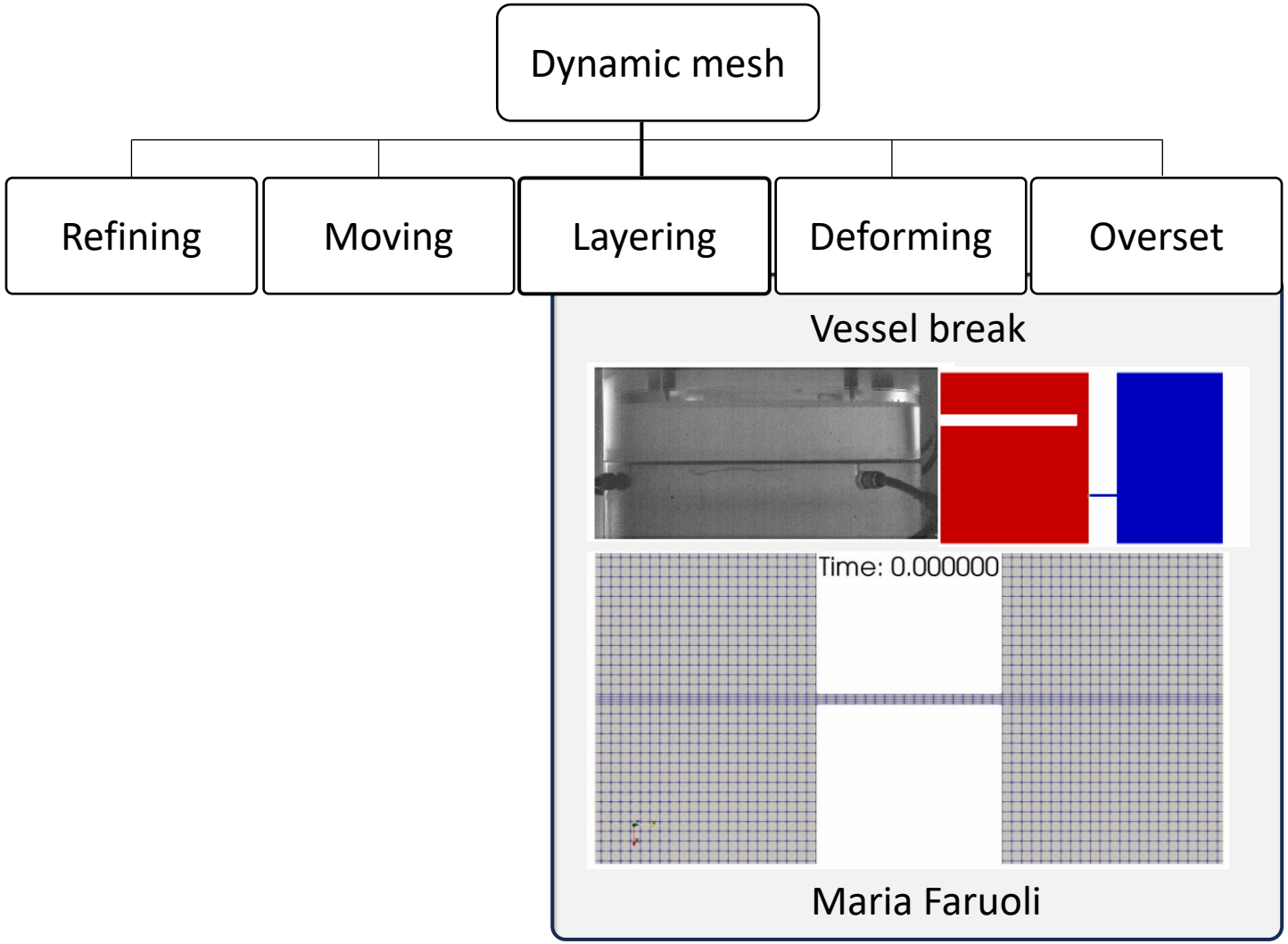
Introduction to the deformable overset method for the simulation of flexible and moving bodies

Maria Faruoli
Romain Poletti
ofcourse@vki.ac.be

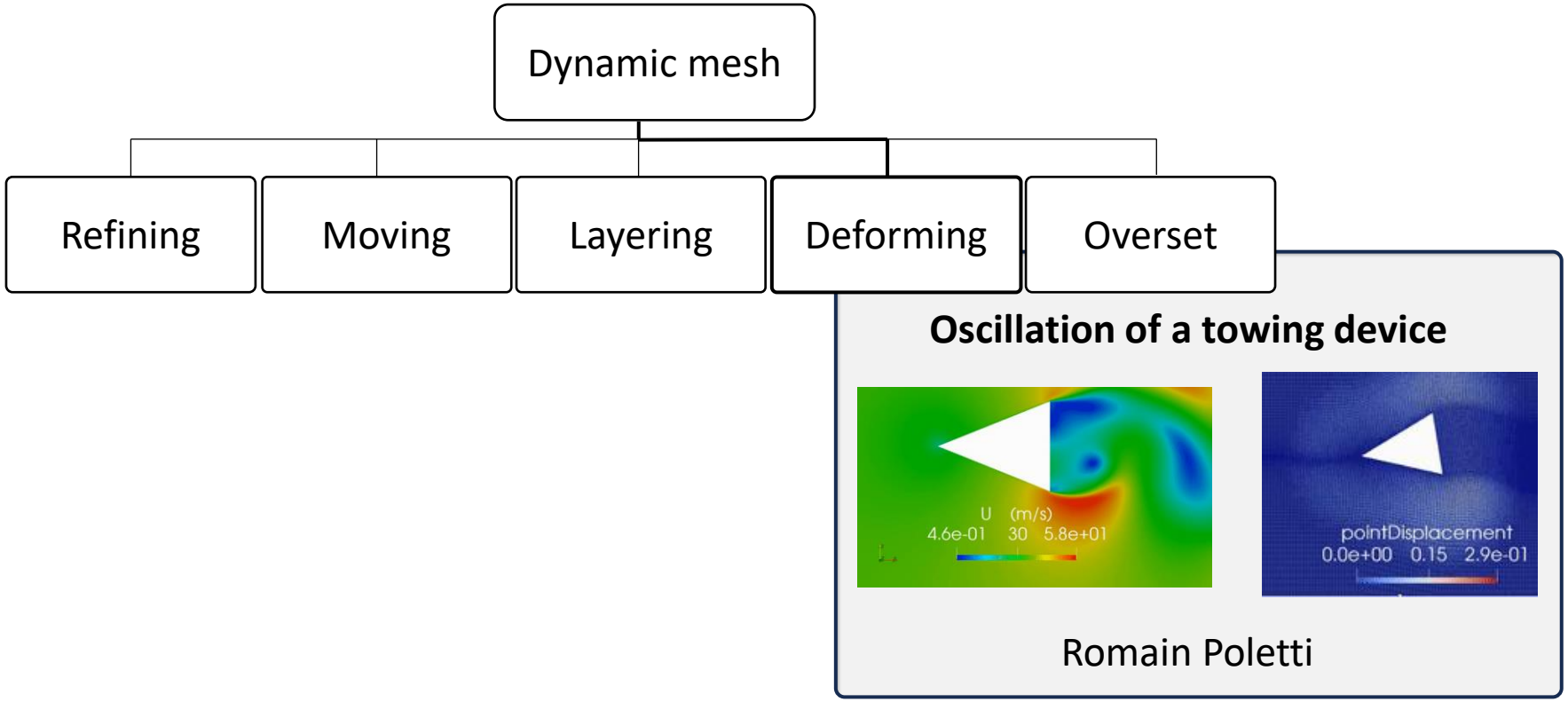
Families of dynamic mesh in OpenFOAM



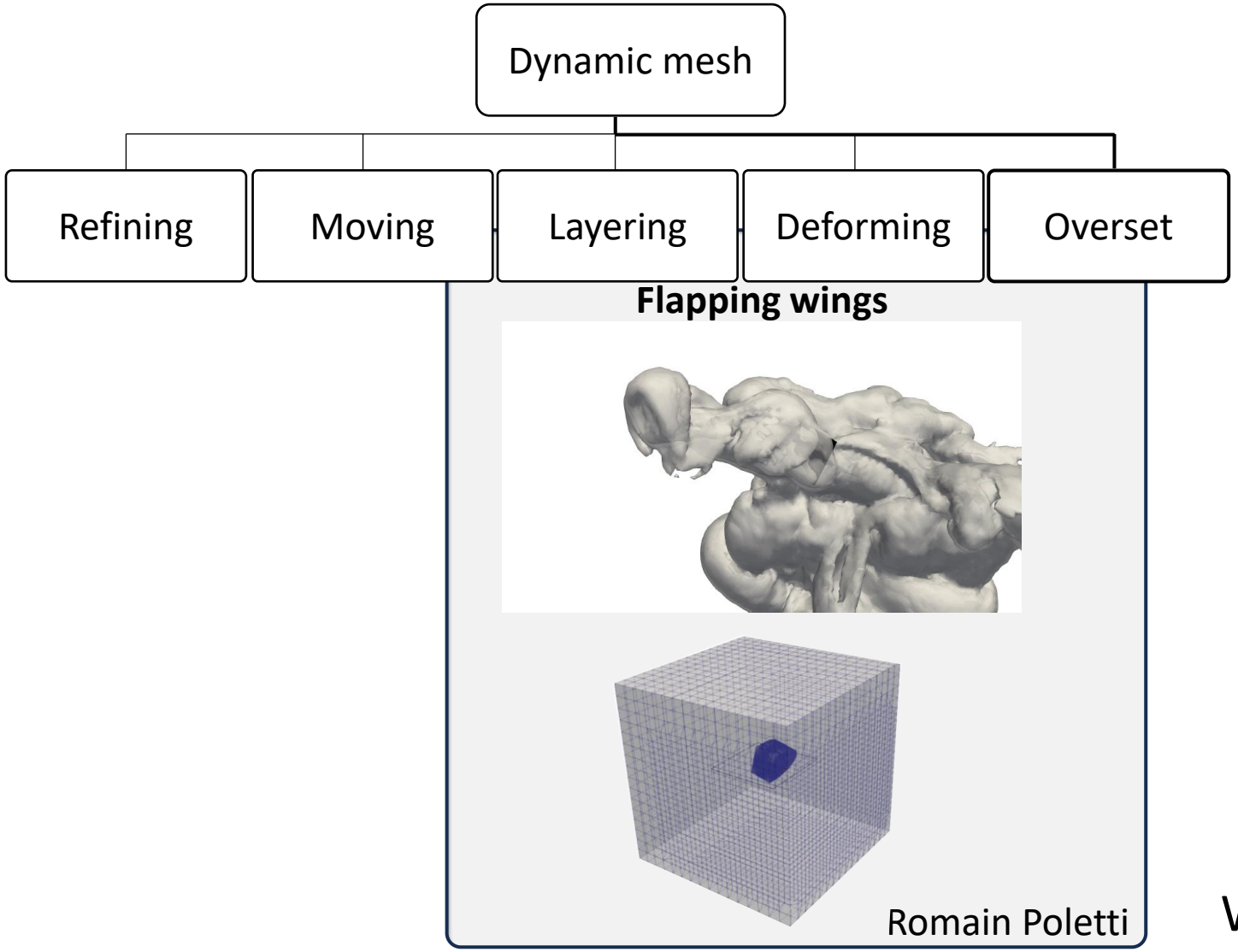
Families of dynamic mesh in OpenFOAM



Families of dynamic mesh in OpenFOAM



Families of dynamic mesh in OpenFOAM



and some other techniques which include:

- Sliding mesh
- Layering with interface
- …

What are **you** working on?

Objective of the tutorial

At the end of this tutorial, you will:

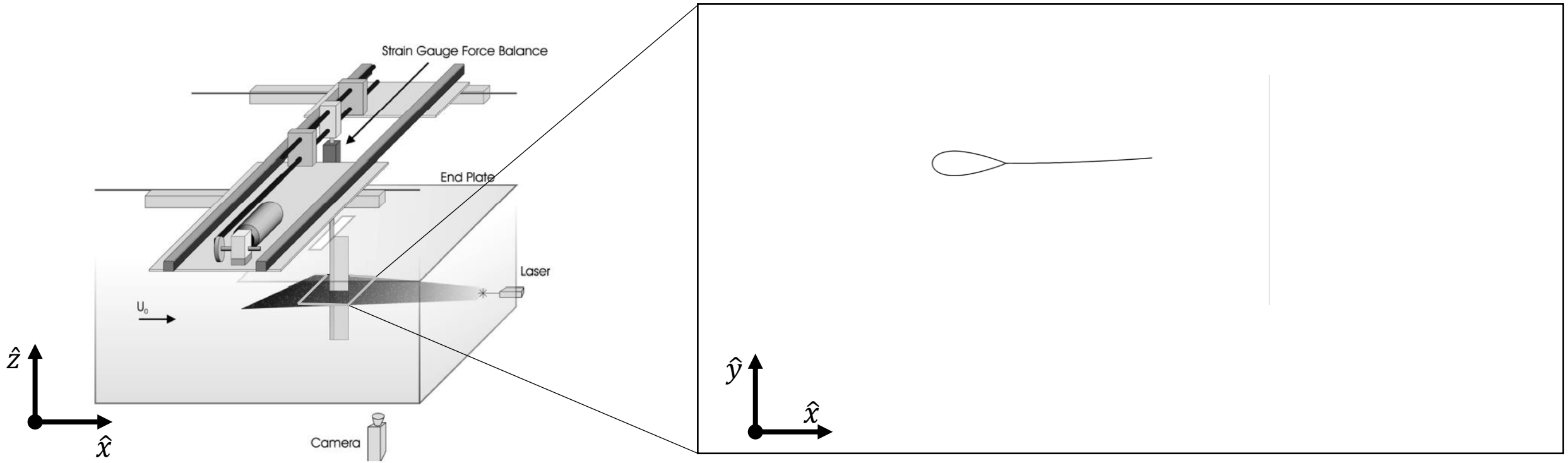
- ✓ Master the different steps to run an overset-based simulation
  - ✓ Understand the basics of the deformable overset method
  - ✓ Master the case set-up
  - ✓ Implement new boundary conditions
  - ✓ Play with basic post-processing tools

- × Master the details working principle of the overset method and its implementation in implementation

Table of content

Test case

A flexible airfoil heaving in water*:



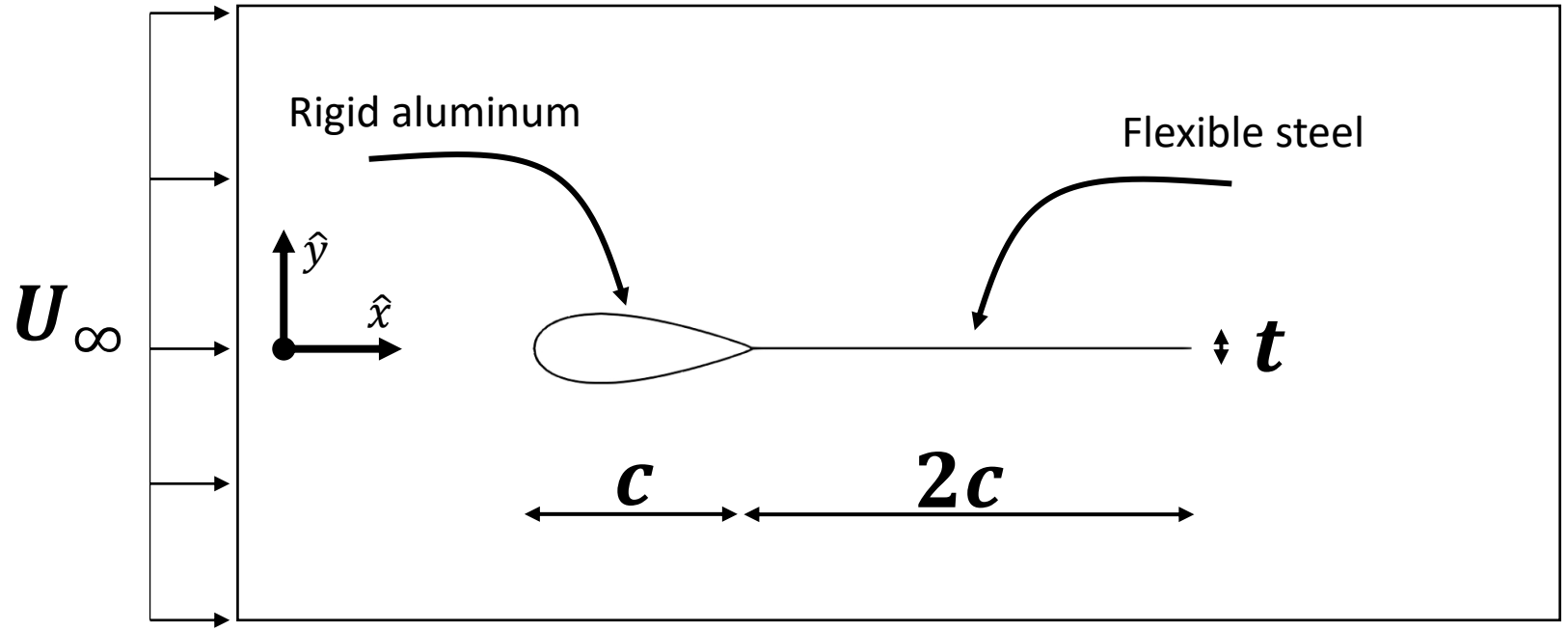
Flapping parameters

$$y(t) = h_a sin(2\pi f t)$$

h_a	1.75 cm
f	0.96 Hz

*Heathcote, S., & Gursul, I. (2007). Flexible flapping airfoil propulsion at low Reynolds numbers. *AIAA journal*, *45*(5), 1066-1079.

Test case



Domain parameters

U_∞	0.1 m/s
ρ_w	1000 kg/m^3
$3c$	0.09 m
t	5e-5 m

Dimensionless parameters

$$Re = \frac{3cU_{ref}}{\nu}$$

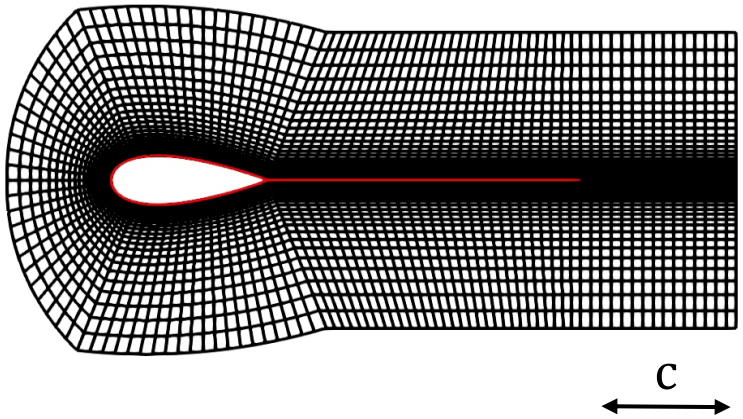
$$St = \frac{2fh_a}{U_{ref}}$$

Re	9k
St	0.34

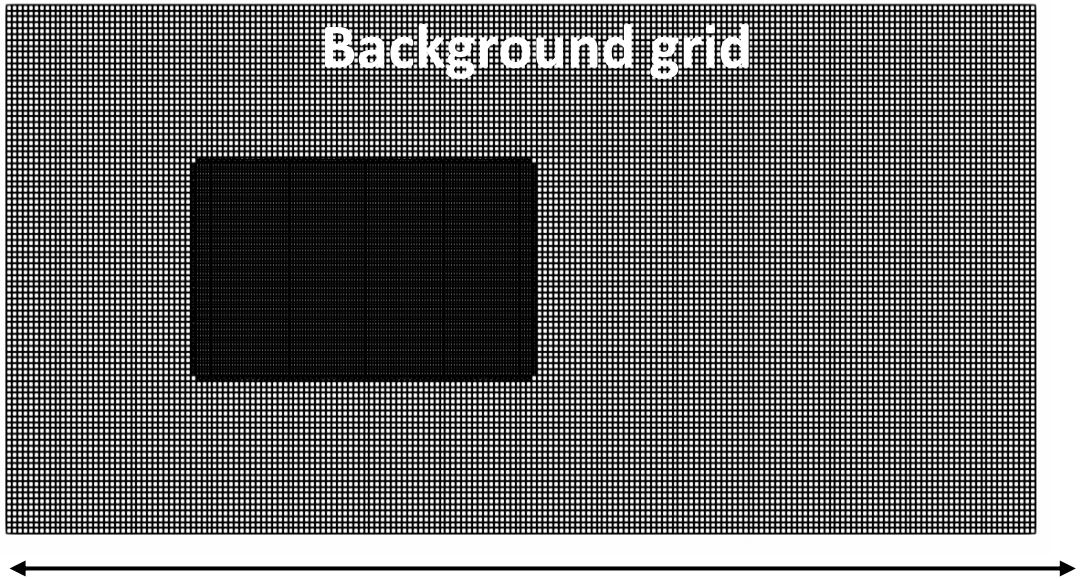
Table of content

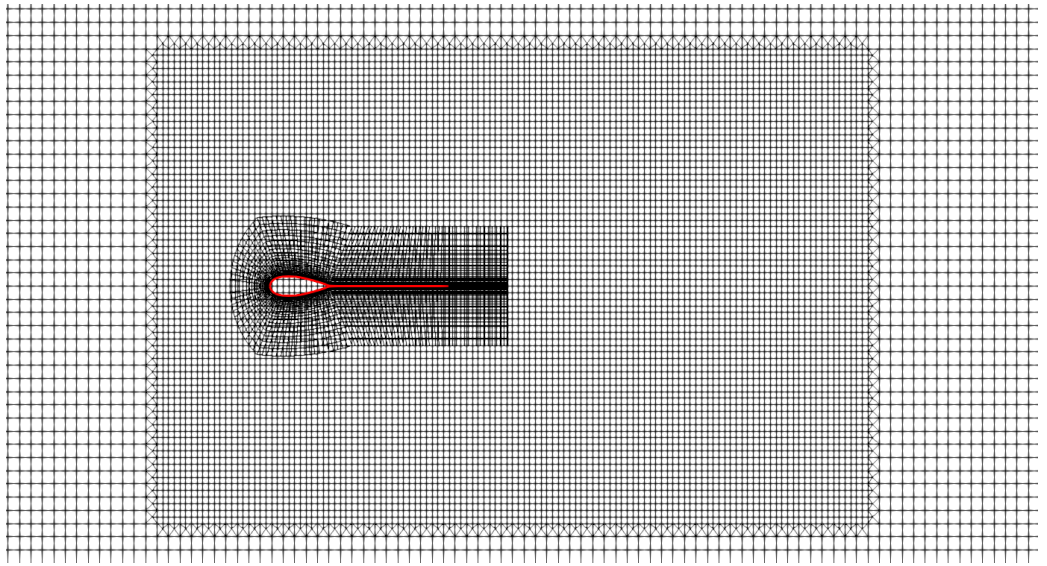
The overset meshing technique (1/2)

Component grid



+

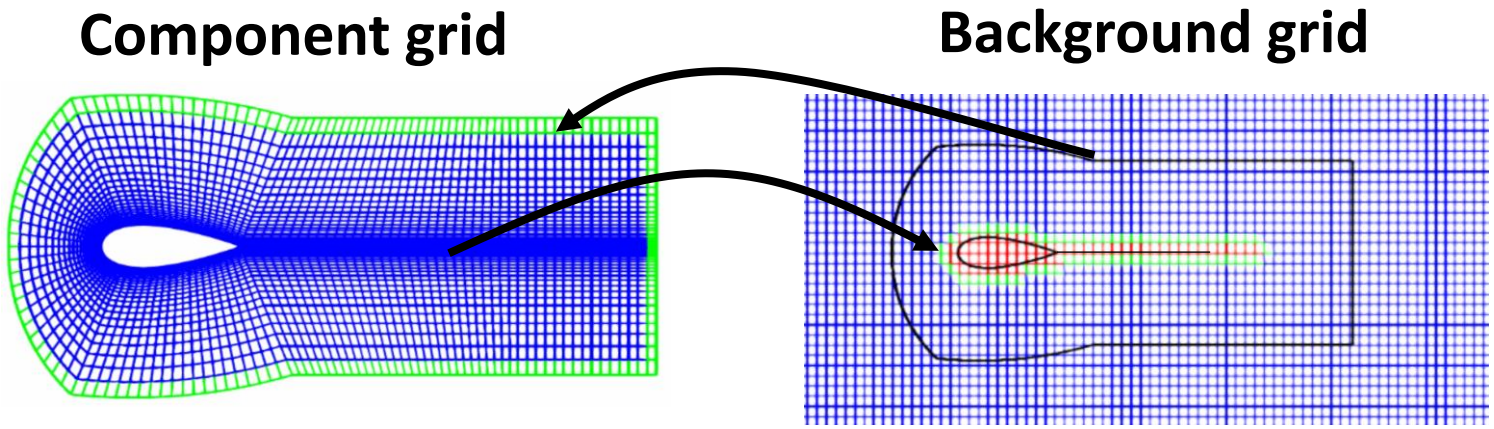




- ✓ No grid match, No re-meshing
- ✓ Accurate flow capture near walls

- × Computational time
- × **Only rigid motion**

The overset meshing technique* (2/2)



Calculated cells

Flow equations are solved

"Normal" cells

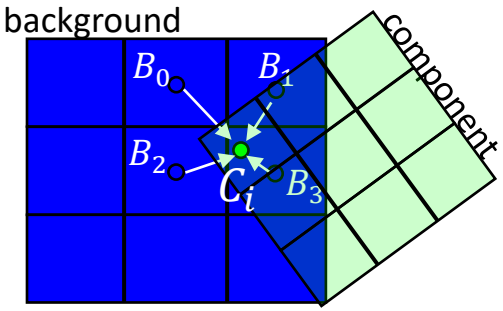
Donor cells

Inactive cells

No equations must be solved

Interpolated cells

Interpolation equations replace the flow equations



Weighted average

$$\phi_{C_i} = \sum_{k=1}^{N_D} w_k \phi_{B_k}$$

Where w_k are weights defined by the interpolation scheme (e.g. inverse distance) [3]

***More theory?** See [2] (general theory) and [3] (implementation in OF)

Table of content

Preliminaries

This is a step-by-step tutorial in which you are asked to fill in partially completed files.
The tutorial is done in OpenFOAM® v2012

- The files are available here:
  - git clone https://github.com/ofcourse-VKI/ofseminar.git
- In this presentation, the following colour code is used:

```
$ line_you_have_to_type_in_the_terminal
```

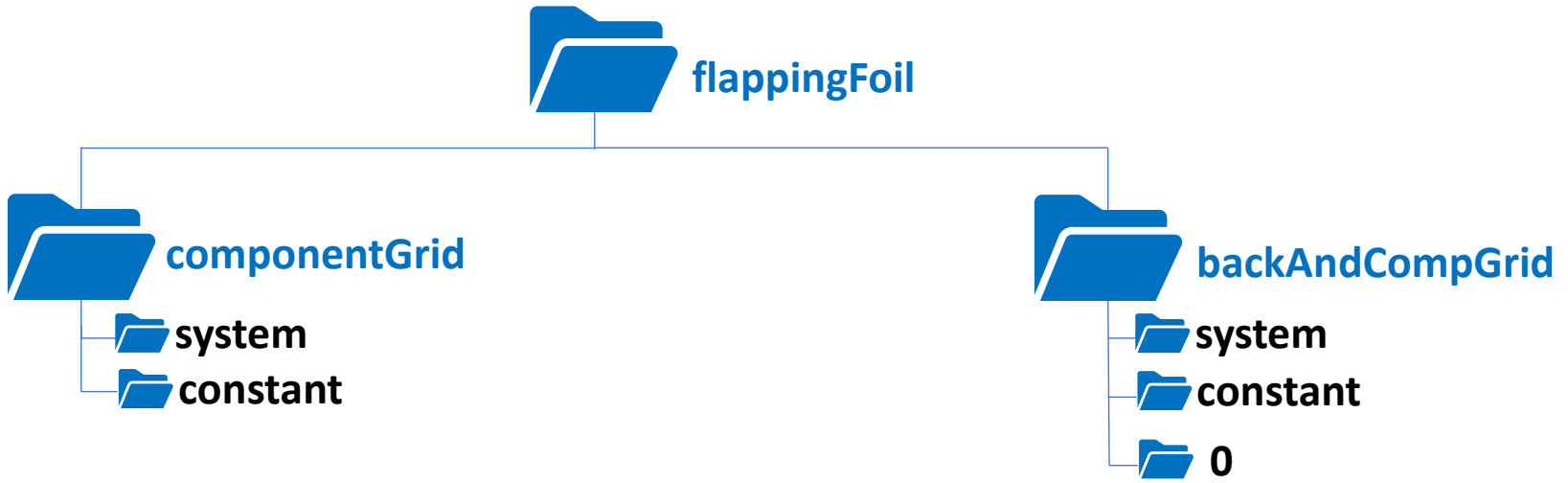
Files you must edit

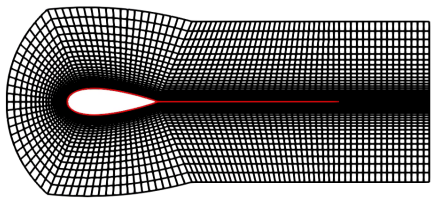
Files you just need to read, not editing needed

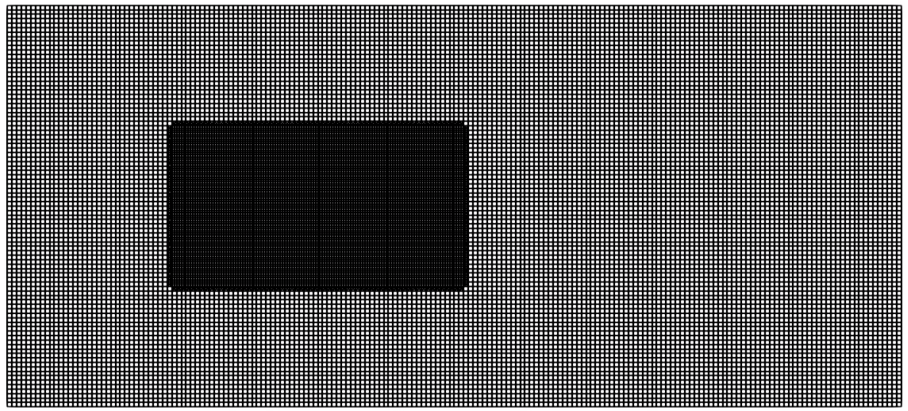
- A file or code snippet is shown with this form:

```
/*
Reference dimensions
*/
c 30.0;              // chord length
l 60.0;              // plate length
(...)

// Number of cells and grading
Ny1 30;
(...)
```

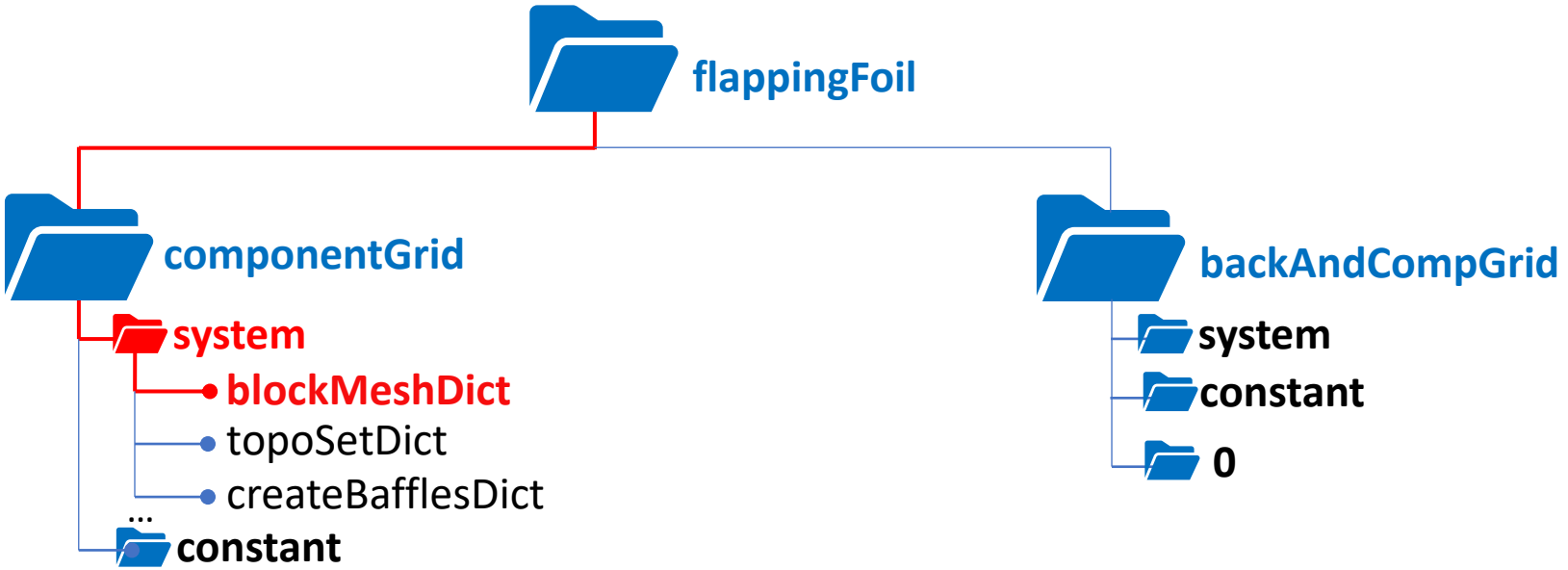
Roadmap


flappingFoil
componentGrid
system
constant
backAndCompGrid
system
constant
0





Roadmap



Let’s start by generating the component grid…

Component grid generation: blockMesh

Let's first have a look at the blockMeshDict, **no need to edit anything**:

```
$ cd componentGrid
$ vim system/blockMeshDict
```

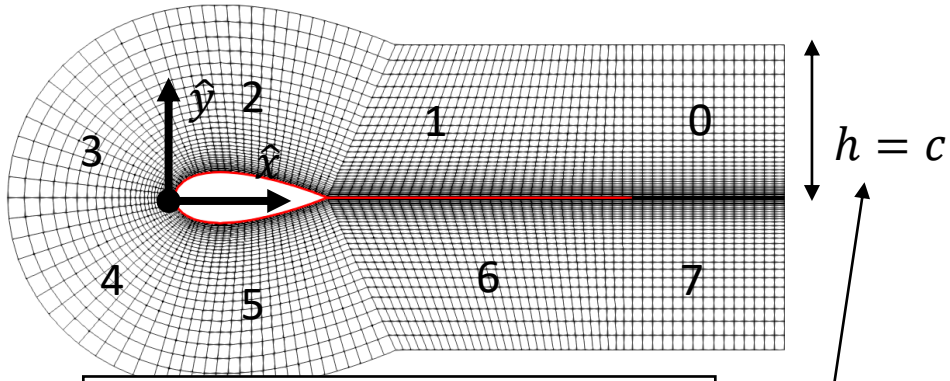
```
1  /*
2  Reference dimensions
3  */
4  c 30.0;                  // chord length
5  l 60.0;                  // plate length
6  (...)
7
8  // Number of cells and grading
9  Ny1 30;
10 (...)
11
12 vertices
13 (
14    ($x2 $y0 $z0) // 0
15    (...)
16 );
17
18 blocks
19 (
20   // block 0 to 8
21   hex (19 20 1 0 27 26 10 9)  movingZone ($Nx1 $Ny1 1)
22   simpleGrading (1 $n1 1)
23   (...)
24 );
25
```

① All the dimensions used by the vertices

② Number of cells and grading for each block

③ Points of the blocks

④ Block definition using ② and ③



To minimize the interpolation errors, h must be large so that the component boundary and its attached interpolated cells are far from the strongest gradient

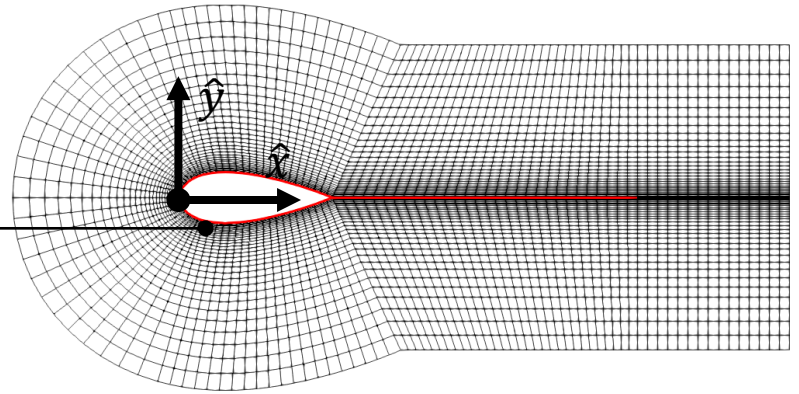
Component grid generation: blockMesh

Let's first have a look at the blockMeshDict, **no need to edit anything**:

```
$ cd componentGrid
$ vim system/blockMeshDict
```

```
edges
(
// Edges of the component boundary around the airfoil
#include "edges/xx.txt"
(...)
// Edges of the airfoil profile
#include "edges/xxx.txt"
(...)
);

```

See plotAirfoil.py



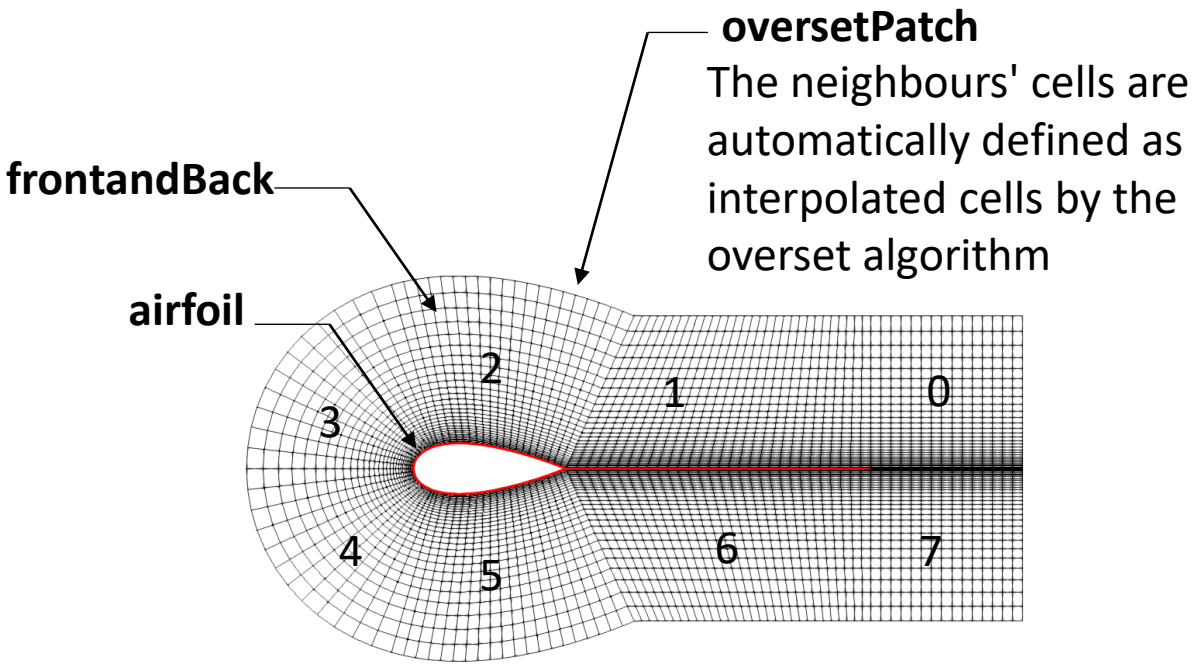
NACA0012: $y = \frac{T}{0.2\,c}\left[a_0\left(\frac{x}{c}\right)^{0,5} + a_1\frac{x}{c} + a_2\left(\frac{x}{c}\right)^2 + a_3\left(\frac{x}{c}\right)^3 + a_4\left(\frac{x}{c}\right)^4\right]$

Component grid generation: blockMesh

Let’s first have a look at the blockMeshDict, **no need to edit anything**:

```
$ cd componentGrid
$ vim system/blockMeshDict
```

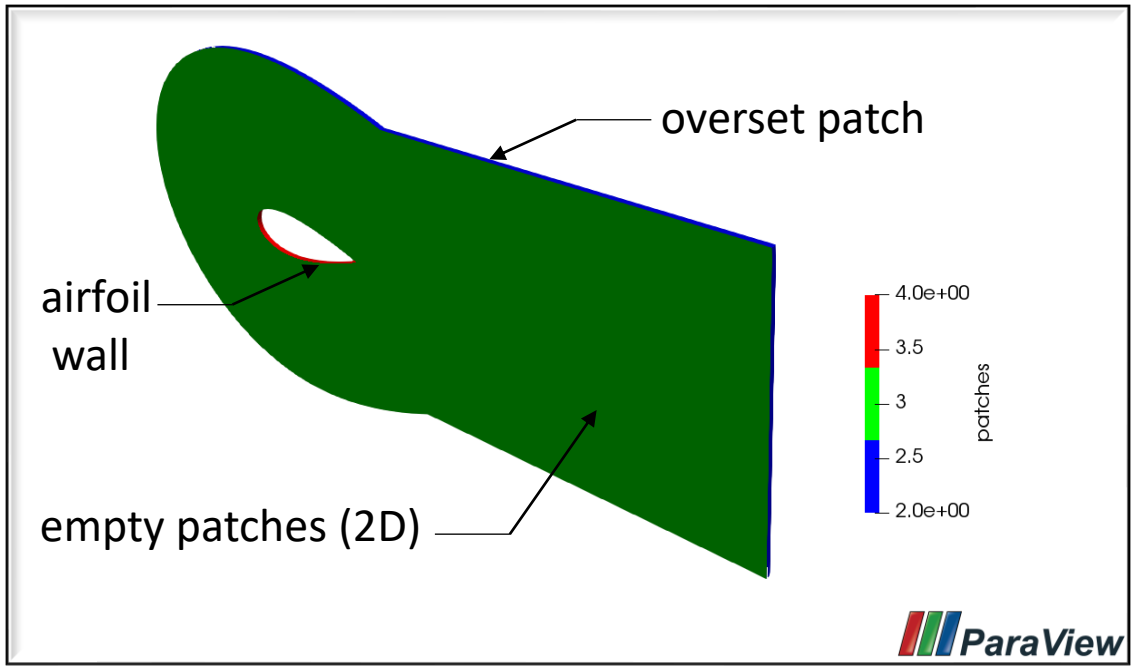
```
boundary
(
    oversetPatch
    {
        type overset;
        faces ((3 2 12 11)
               (...));
    }

    frontAndBack
    {
        type empty;
        faces (...);
    }
    airFoil
      {
        type wall;
        faces(...)
      }
    }
```



Component grid generation: blockMesh

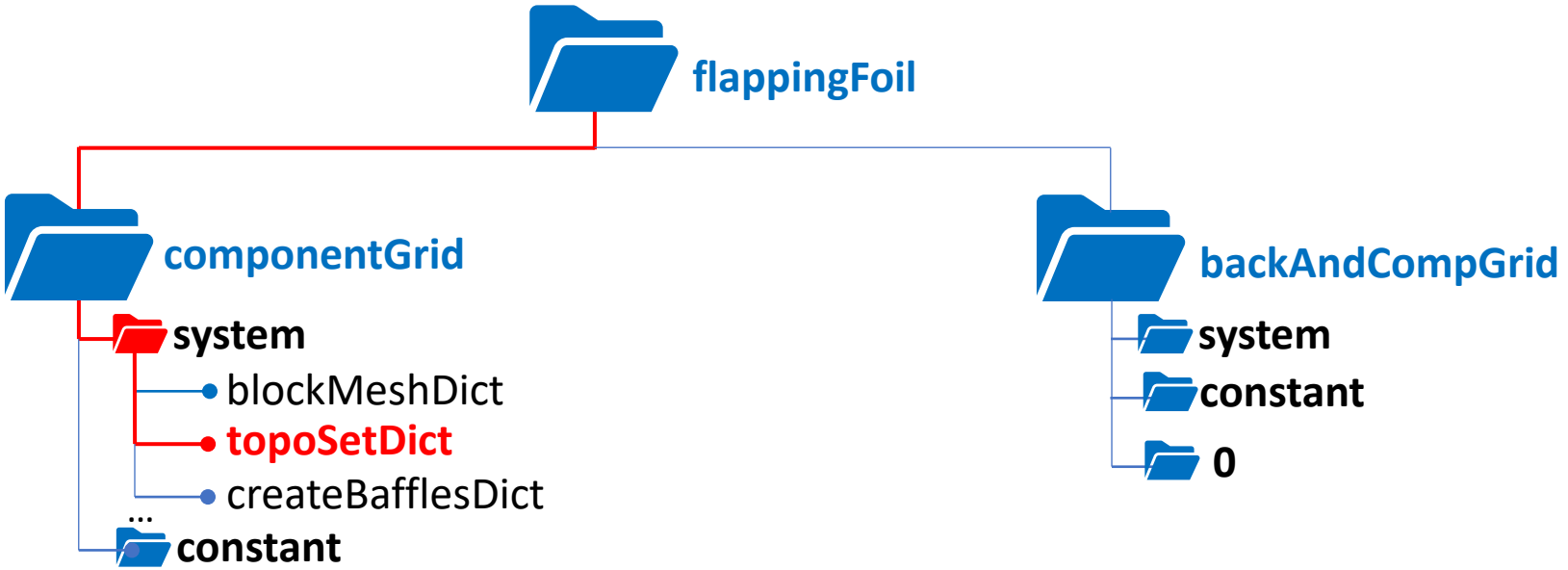
Let’s run blockMesh and open paraview:

```
$ blockMesh
$ paraFoam
```



The plate is not defined yet!

Roadmap



The airfoil walls were defined, we need to define the plate…

Definition of the plate (1/2): toposetDict

Hypotheses: the plate thickness is negligible: $\frac{t}{c} \ll 1$. **We don't model the plate thickness.**

Advantages: Easy mesh definition in blockMesh & decrease of the number of cells

Methodology to model the plate:

- **A baffle is used to model the plate:** internal faces of zero thickness that can hold a boundary condition.
- First **open toposetDict** to define the location of the baffle as a face zone:

```
actions
(
// 1. Create a set of faces (faceSet) that contains
// all the faces that form the wing's plate
      {
        name    //!FILL HERE! // Define a name
        type    //!FILL HERE! // Define the type
        action  new;          // new set
        source  boxToFace;    // Create a thin box that include the plate locatio
        sourceInfo
        {
           // Define the extreme points of the box
          // !FILL HERE!  // e.g. box (xmin ymin zmin) (xmax ymax zmax)
        }
      }

// 2. Convert this faceSet into a faceZoneSet
       {
         name    //!FILL HERE! // Define a name
         type    //!FILL HERE! // Define the type
         action  new;
         source  setToFaceZone;
         sourceInfo
         {
           // Use the type and name of the faceSet defined in the previous action
           // !FILL HERE!
         }
       }
);
```

Definition of the plate (1/2): toposetDict

Hypotheses: the plate thickness is negligible: $\frac{t}{c} \ll 1$. We don't model the plate thickness.

Advantages: Easy mesh definition in blockMesh & decrease of the number of cells

Methodology:

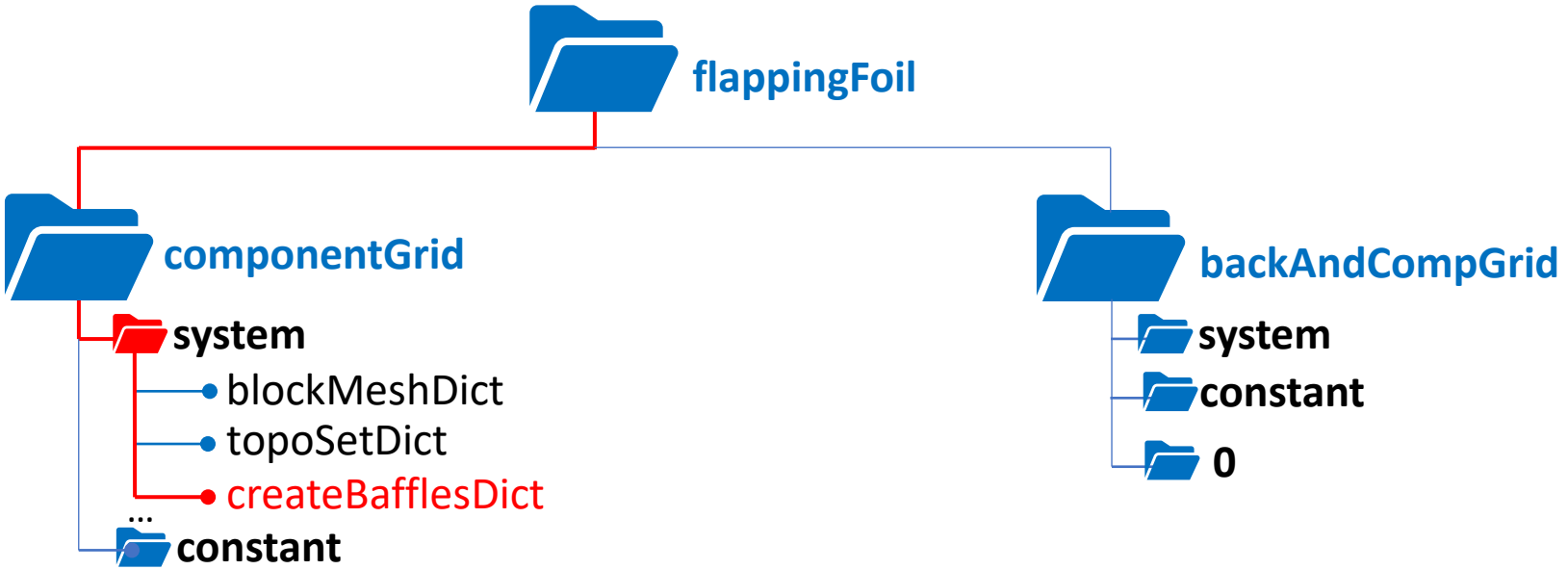
- **A baffle is used to model the plate:** internal face of zero thickness that can hold a boundary condition.
- First **open toposetDict** to define the location of the baffle as a face zone:

```
actions
(
// 1. Create a set of faces that contains all the faces that form the wing's plate
        {
          name    flatPlateFace;
          type    faceSet;
          action  new;
          source  boxToFace;
          sourceInfo
          {
             //extreme points of a thin box that includes the plate location
             box (0.030 0 0) (0.090 0 0.001);
          }
        }

// 2. Convert this faceSet into a faceZone
        {
          name    flatPlate;
          type    faceZoneSet;
          action  new;
          source  setToFaceZone;
          sourceInfo
          {
           faceSet flatPlateFace;
          }
        }
);
```

And run it:

```
$ topoSet
```

Roadmap



The airfoil walls were defined, we need to define the plate…

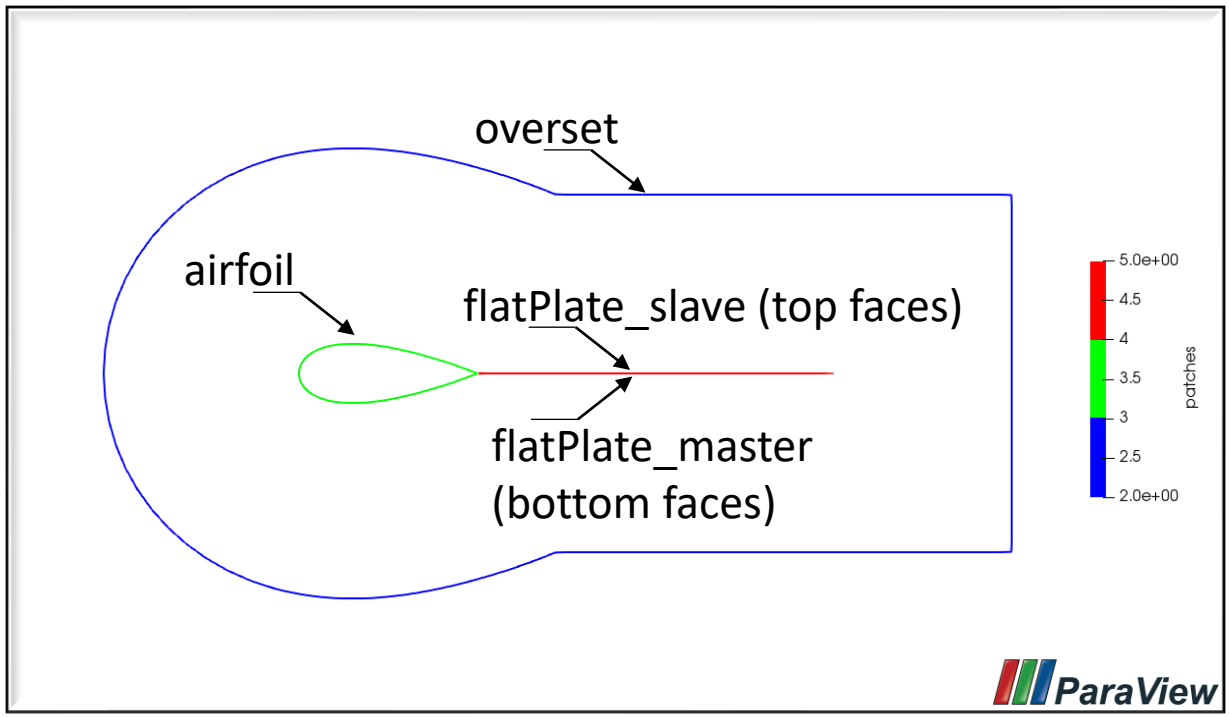
Definition of the plate (2/2): createBaffleDict

Then, **edit createBaffleDict**:

```
internalFacesOnly true;

// Baffles to create.
baffles
{
    flatPlate
    {
        //- Use the faceZone defined with toposetDict
        type        faceZone;
        zoneName    flatPlate;
        patchPairs
        {
            type            wall;
            patchFields
            {
            }
        }}}}
```

Defined in toposetDict

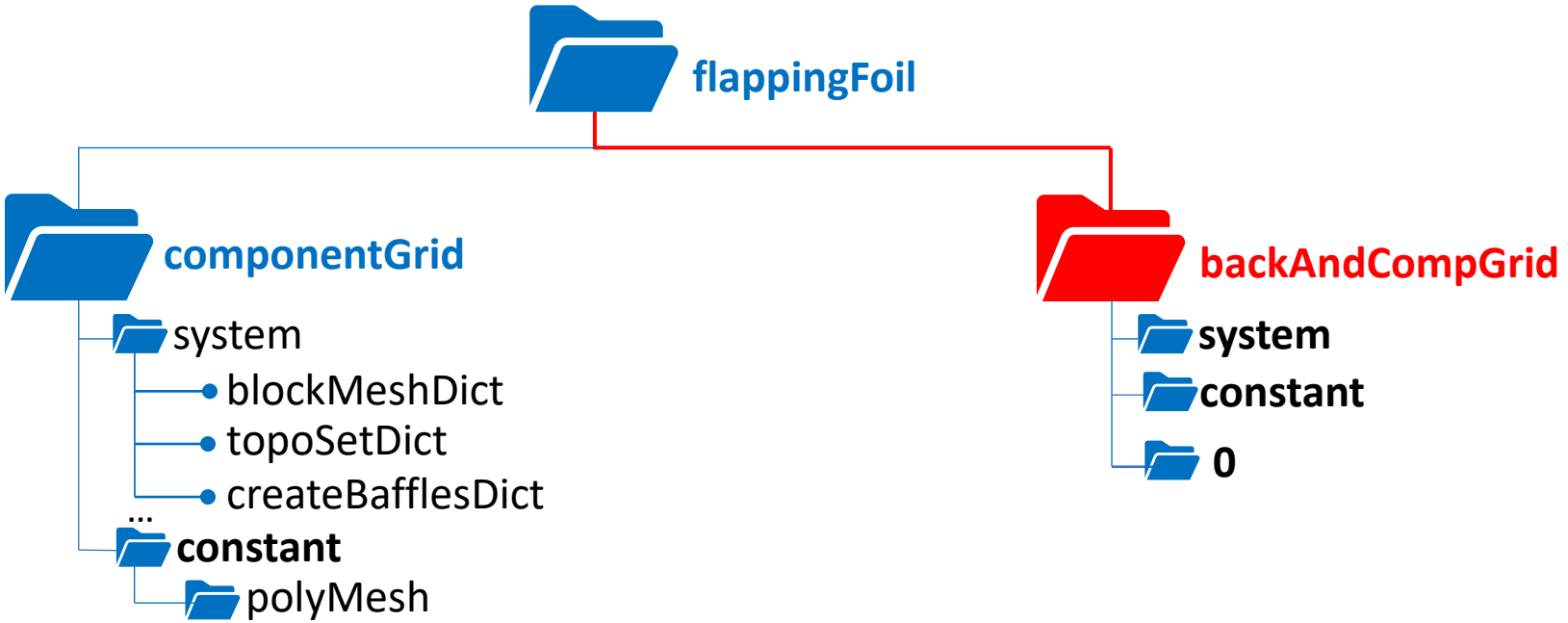
type of the BC



Run it and make sure you have the plate patches defined in Paraview

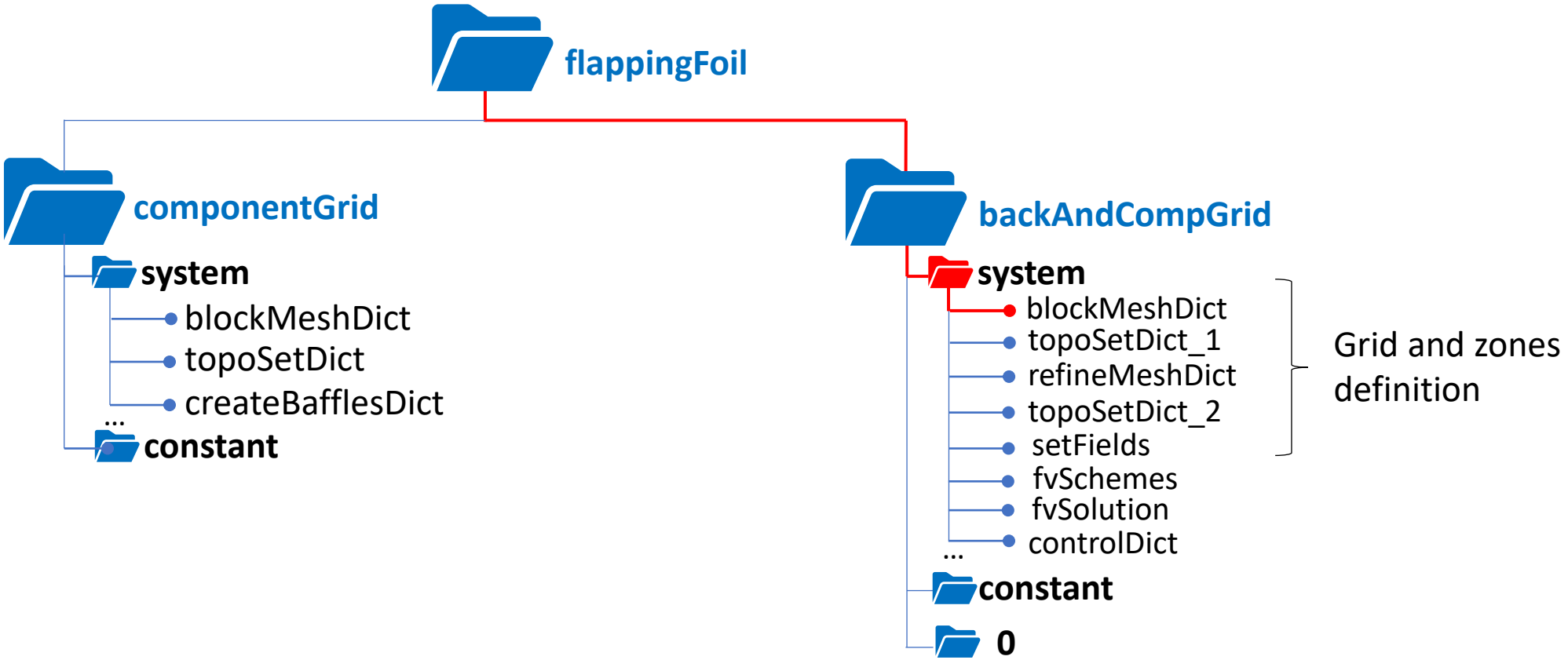
```
$ createBaffles -overwrite
```

Roadmap



The component grid is ready!

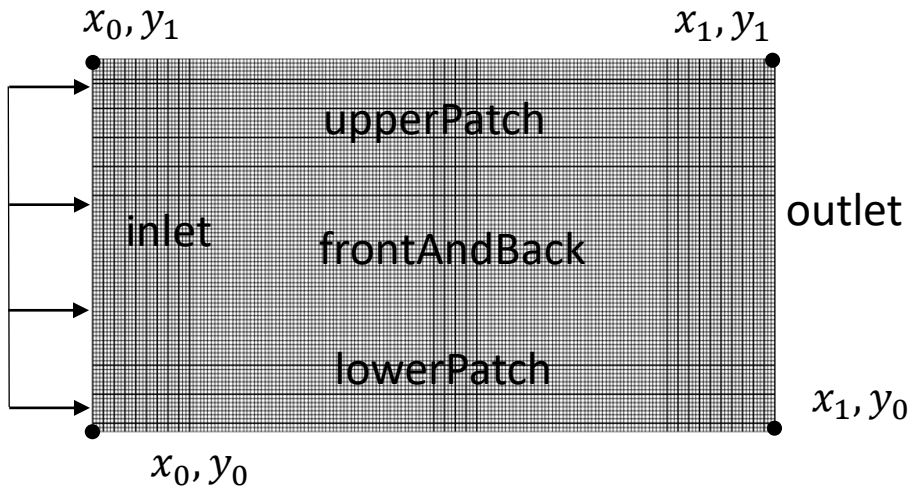
Roadmap



Let’s generate the blockMesh

Background grid (1/2): blockMesh

Open blockMeshDict; a unique block is defined with uniform cells.

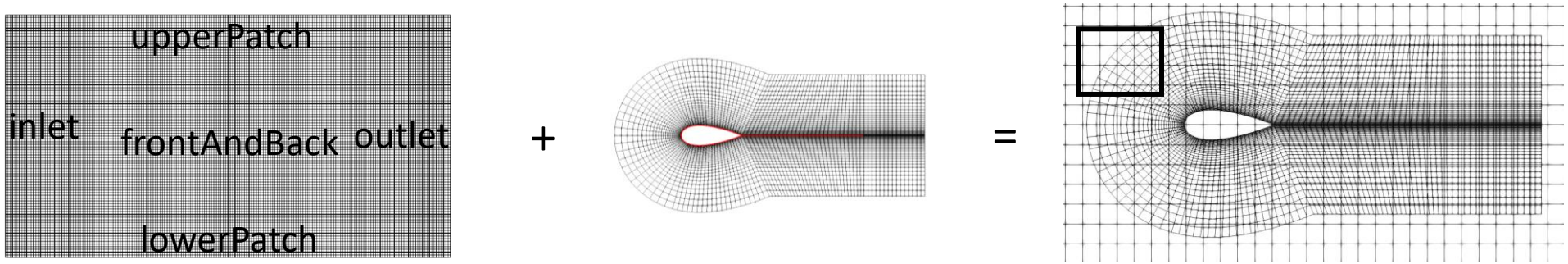


Make sure the coordinates of the domain $x_{0,1}, y_{0,1}, z_{0,1}$ are wisely chosen and generate the mesh:

```
$ blockMesh
```

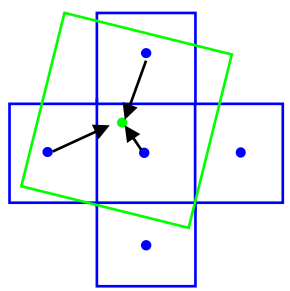
Merging component and background grid ?

Merging?

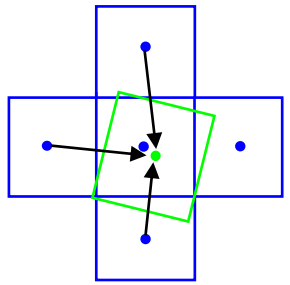


Not yet: The cells of the donors (background) and the interpolated cells (component grid) must have **similar size** to minimize the interpolation errors of the overset method.

background
component

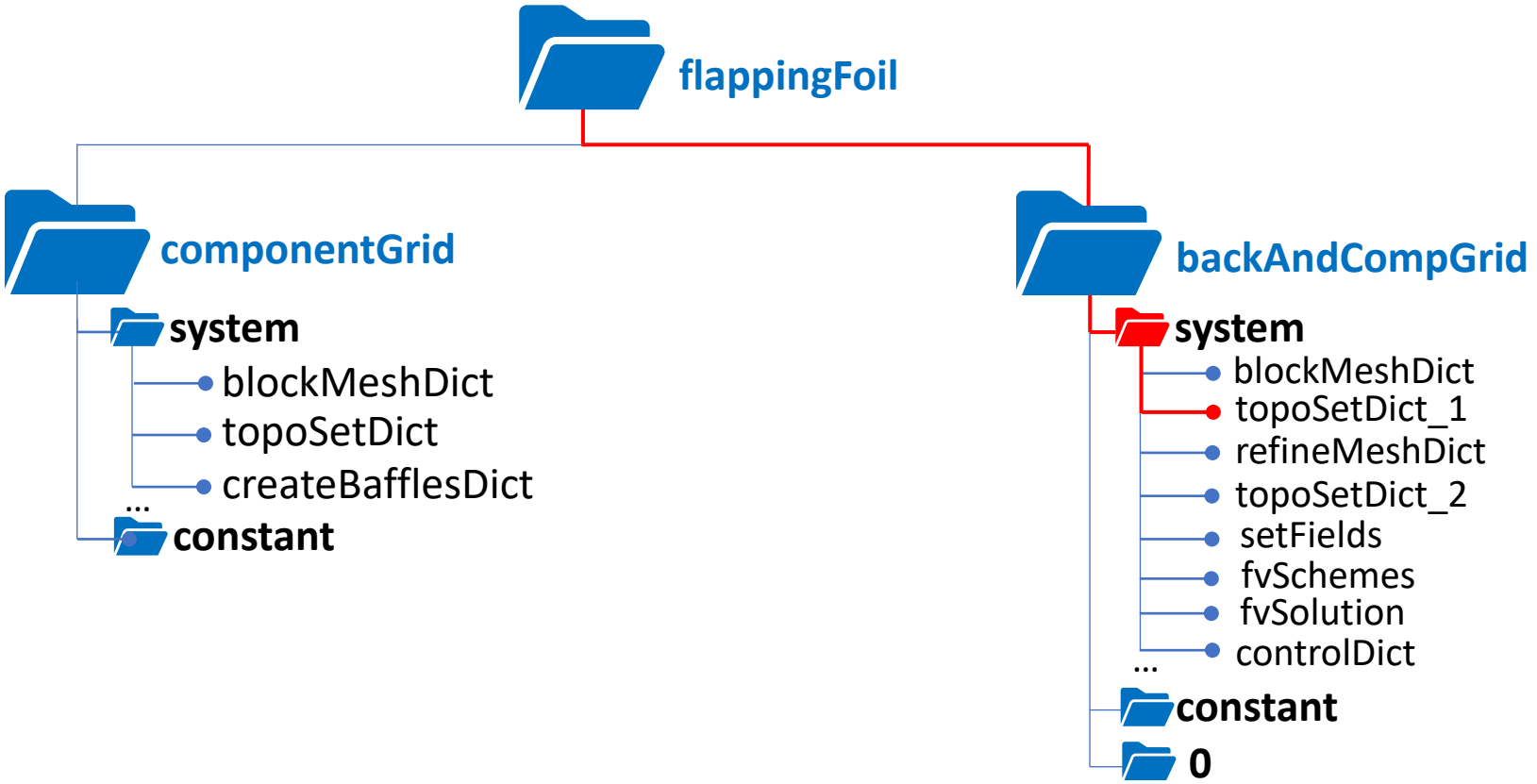


Smear gradients



Optimal

Roadmap



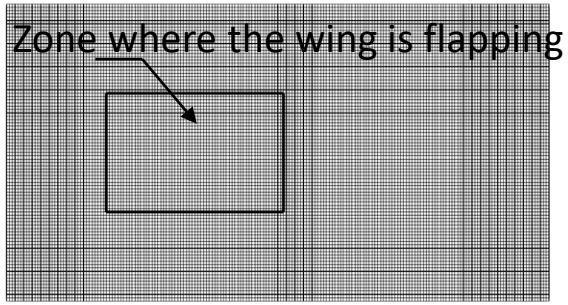
Let’s refine the background grid!

Background grid (2/2): topoSet and refineMesh

So far, the background grid has cells twice bigger than boundary cells of the component grid.
To refine the background grid:

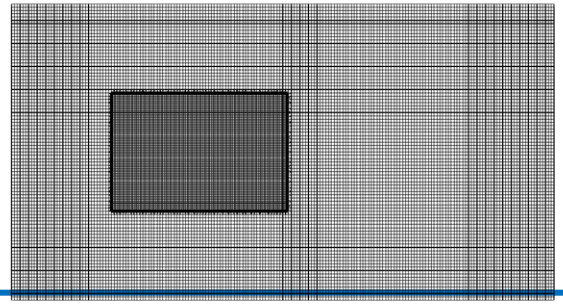
1. **Open topoSetDict_1;** it defines a cell zone to be refined. Then, execute:

```
$ topoSet -dict system/topoSetDict_1
```



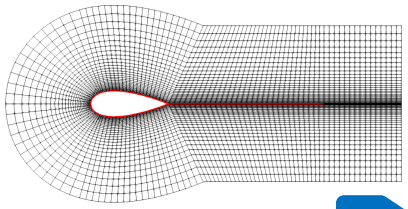
2. Open **refineMeshDict;** use the defined cell zone and set the direction to refine. Then execute:

```
$ refineMesh -overwrite; rm -r 0
```

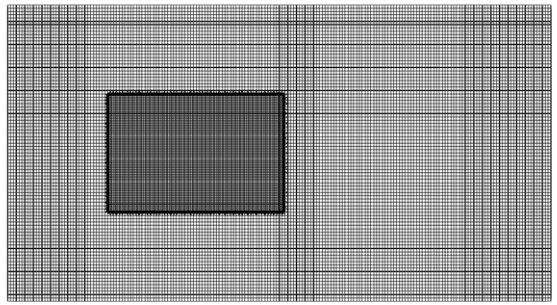


The cell size in the cell zone have been divided by 2

Roadmap



flappingFoil



componentGrid

- system
  - blockMeshDict
  - topoSetDict
  - createBafflesDict
  - ...
- constant

backAndCompGrid

- system
  - blockMeshDict
  - topoSetDict_1
  - refineMeshDict
  - topoSetDict_2
  - setFields
  - fvSchemes
  - fvSolution
  - controlDict
  - ...
- constant
- 0

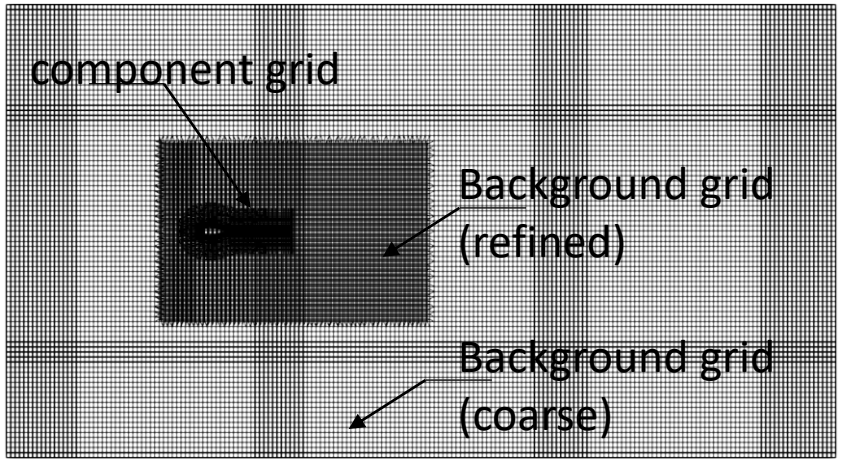
Component grid (CG) is ready!
Background grid (BG) is ready!
Let's merge them!

Grid merging

The **background grid** must now be **merged** with the **component grid**. We have one unique domain with two overlapping grids:

```
$ mergeMeshes    .    ../componentGrid    -overwrite
```

background component

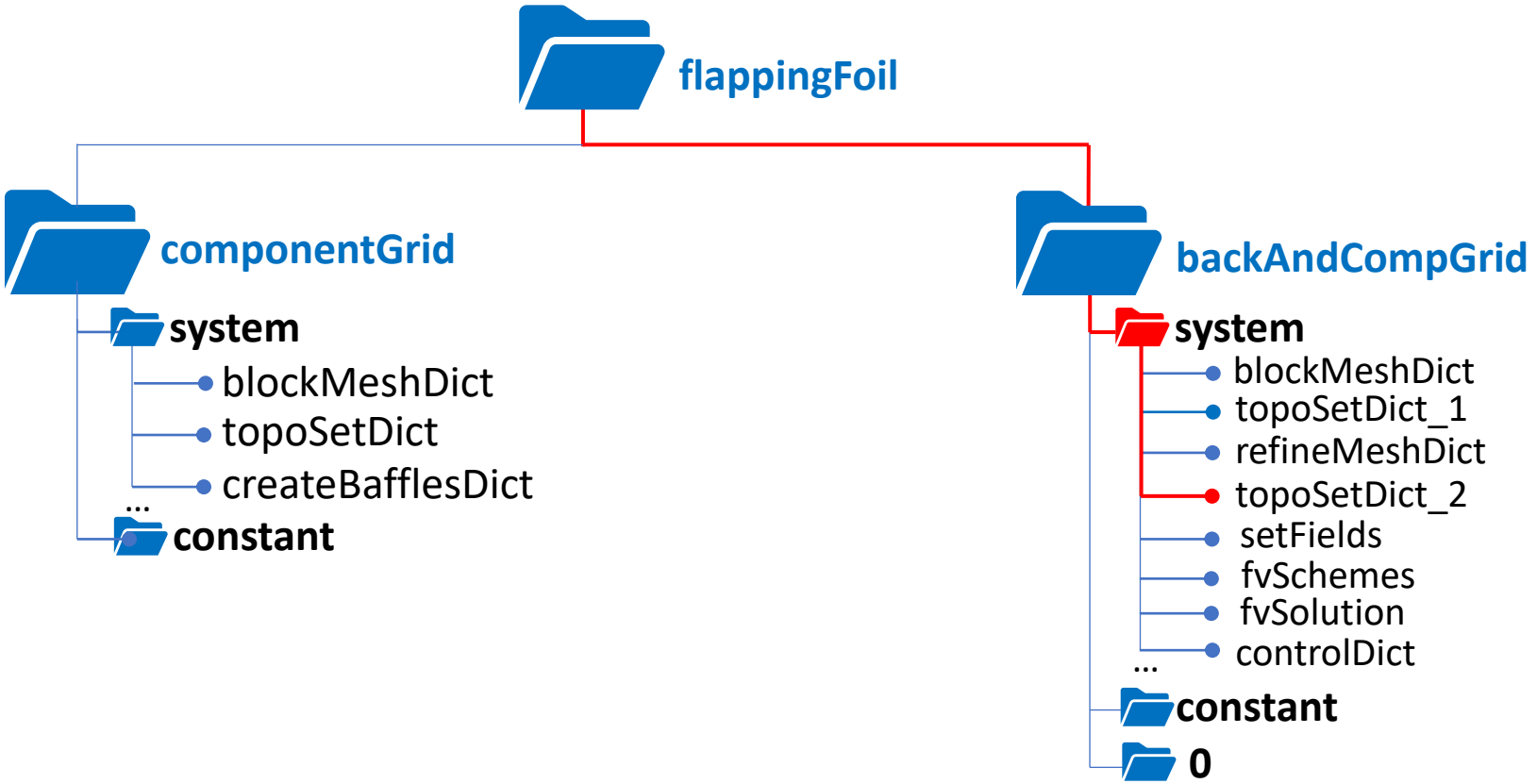


Careful:

- Do add the cells of the component to the background grid
- × Do not merge the faces
- × Do not connect the grids

The algorithm (and the user) need to be able to address the component and the background grid individually

Roadmap



Let’s identify zones of the CG and zones of the BG!

Define mesh zones: topoSetDict

Open and edit topoSetDict_2:

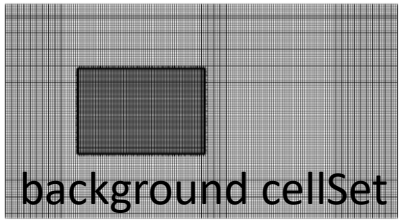
```
{
    name    background;
    type    cellSet;
    action  new;
    source  regionToCell;
    insidePoints ((-0.125 0 0.0005));
}
{
    name    component;
    type    cellSet;
    action  new;
    source  cellToCell;
    set     background;
}
{
    name    component;
    type    cellSet;
    action  invert;
}
{
 name    movingZone;
 type    cellZoneSet;
 action  subset;
 source setToCellZone;
 sourceInfo
 {
    set component;
 }
```

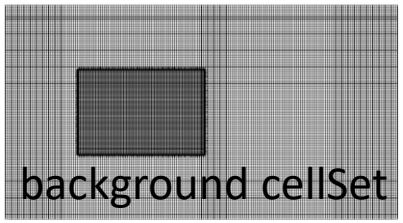
New

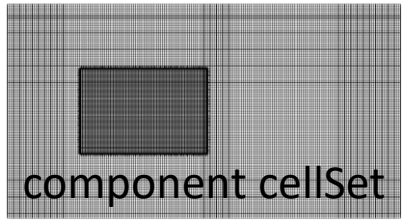
New

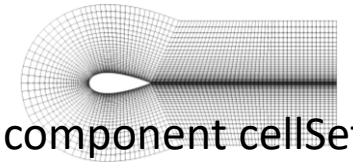
Invert

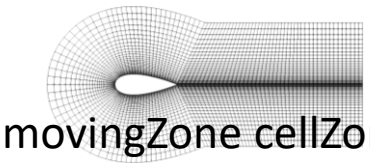
Transform









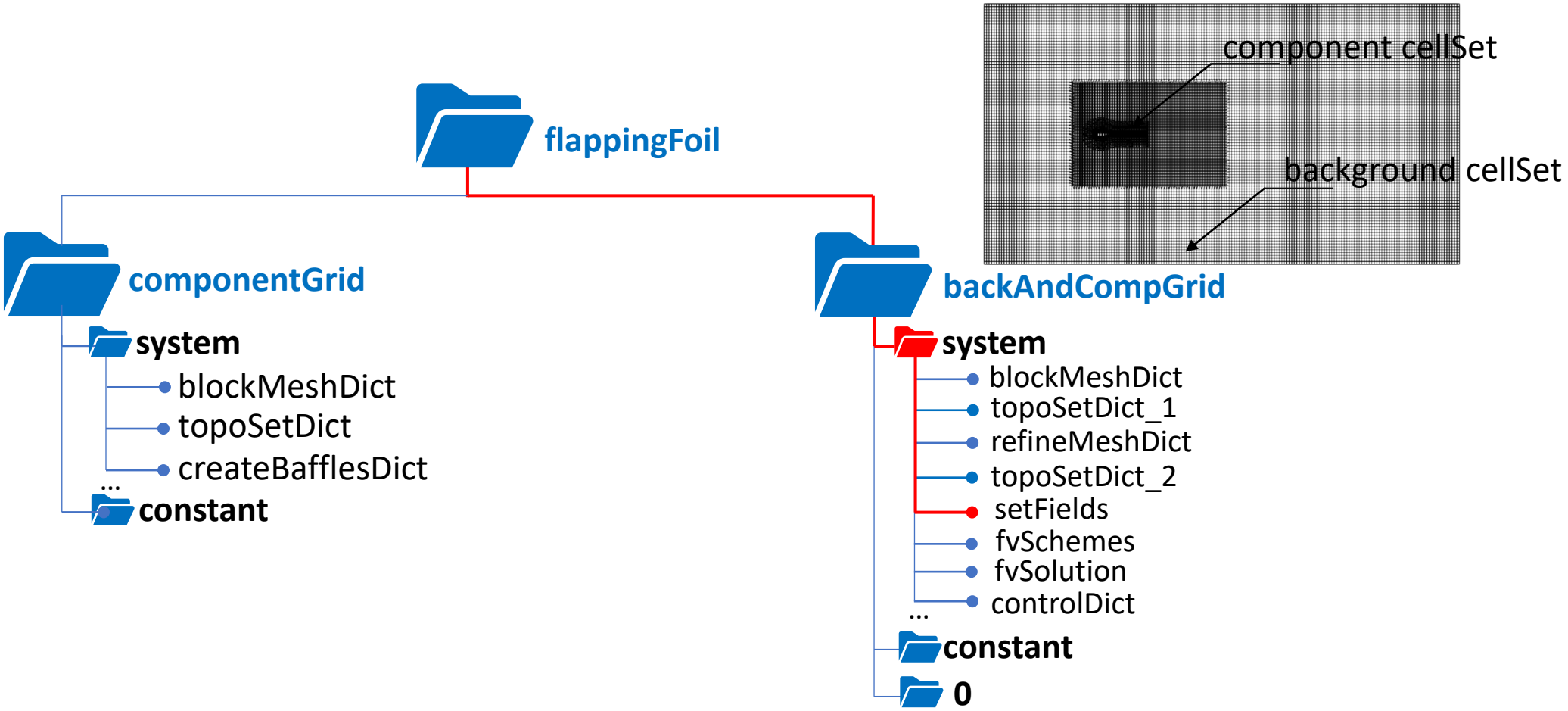


Used by dynamicMeshDict, see *later in the tutorial*

Run toposet:

```
$ topoSet -dict system/topoSetDict_2
```

Roadmap

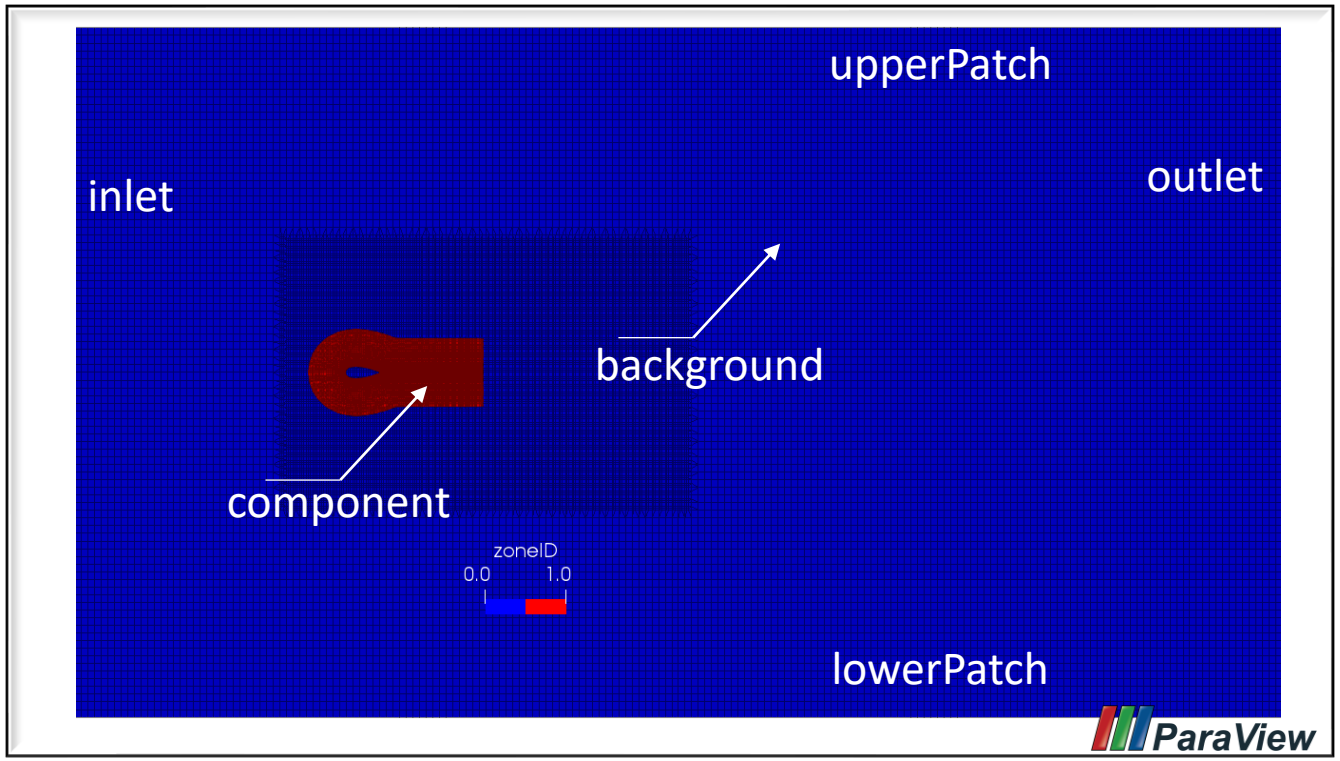

component cellSet
background cellSet
flappingFoil
componentGrid
system
blockMeshDict
topoSetDict
createBafflesDict
...
constant
backAndCompGrid
system
blockMeshDict
topoSetDict_1
refineMeshDict
topoSetDict_2
setFields
fvSchemes
fvSolution
controlDict
...
constant
0

Define zoneID

The cell sets defined with topoSetDict are given an ID (zoneID) that are used by the solver to address the different zones.

Open setFieldsDict:

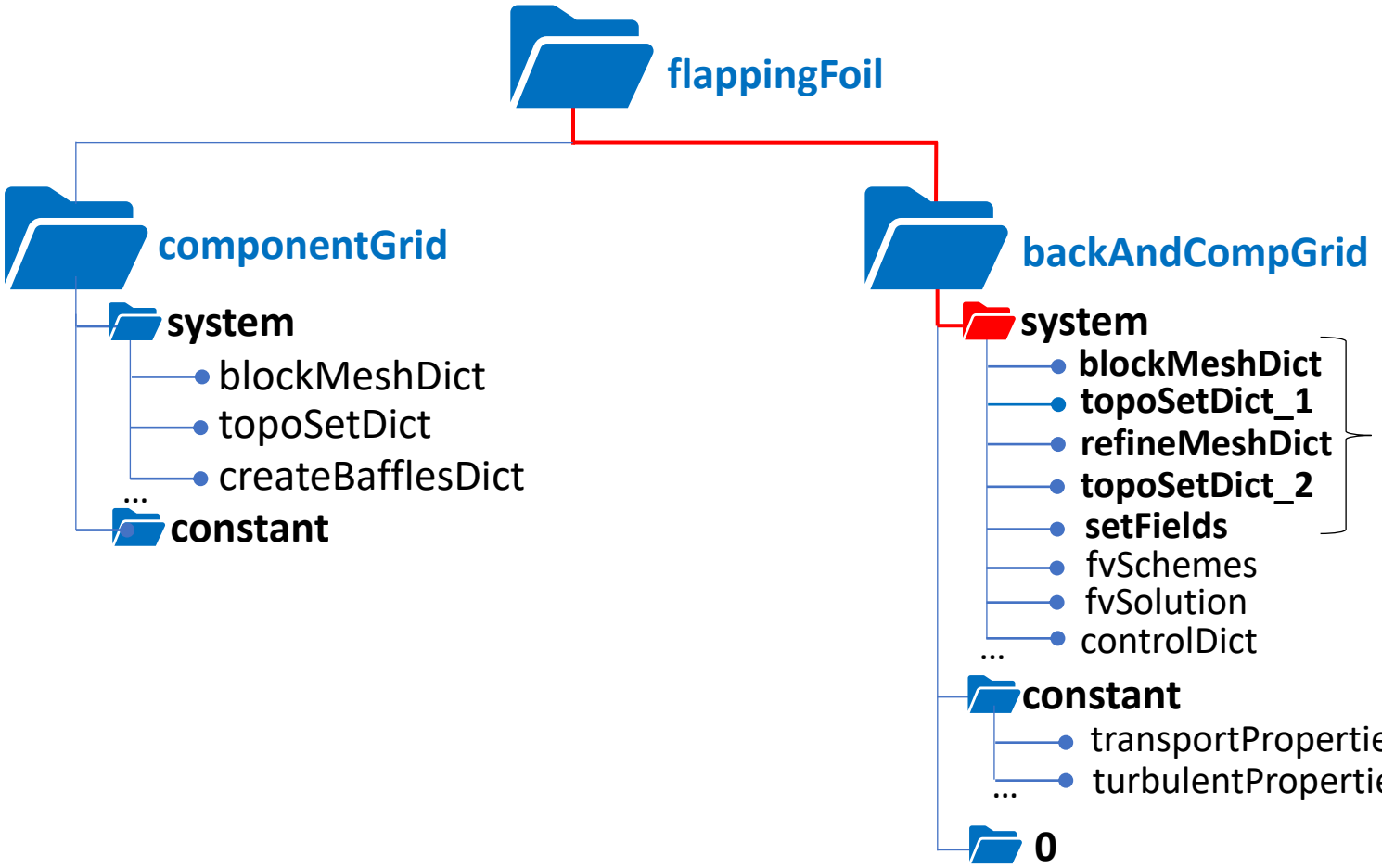
```
defaultFieldValues
(
    volScalarFieldValue zoneID 0
);
regions
(
    cellToCell
    {
        set component;

        fieldValues
        (
            volScalarFieldValue zoneID 1
        );
    }
    cellToCell
    {
        set background;

        fieldValues
        (
            volScalarFieldValue zoneID 0
        );
    }
);
```



Run it an open Paraview to visualize the zoneID field:

```
$ cp -r 0.orig 0; setFields
```

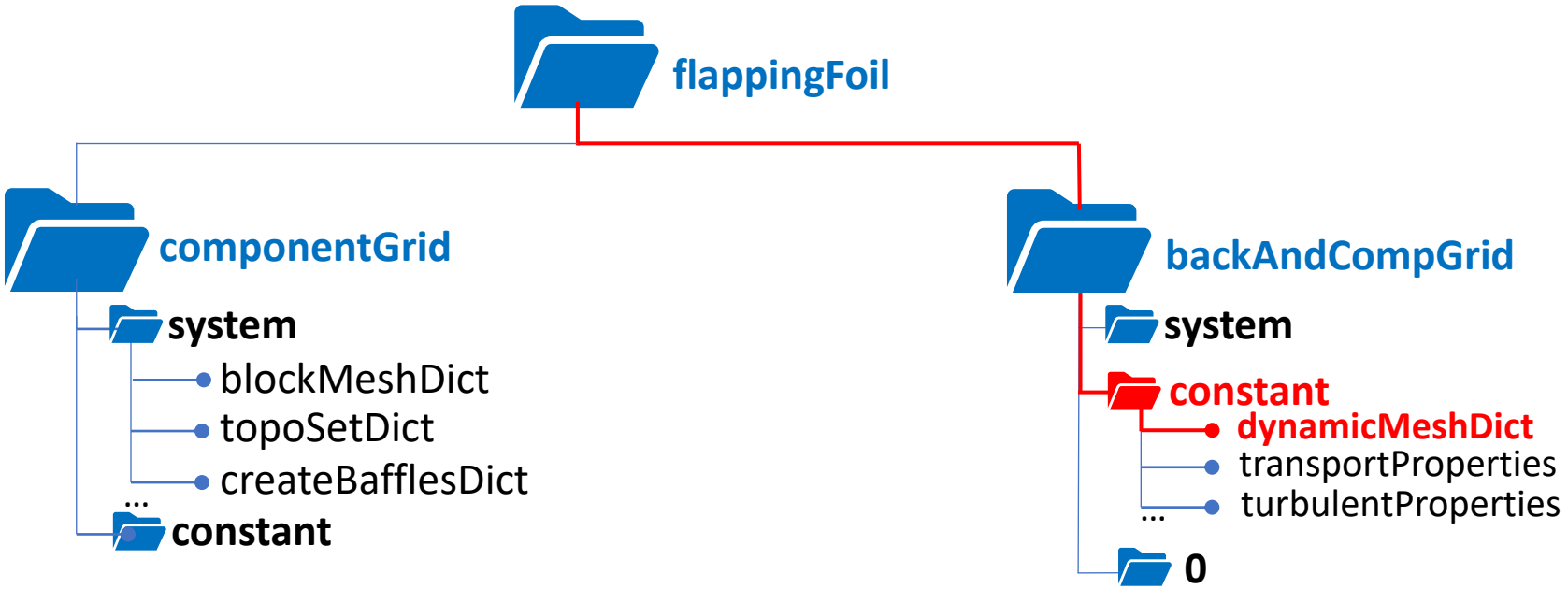
Roadmap



So far:

1. Generate a uniform background grid
2. Identify a zone of refinement
3. Refine there
4. Merge CG and BG
5. Identify BG and CG set
6. Set an ID to those sets

Roadmap


flappingFoil
componentGrid
system
blockMeshDict
topoSetDict
createBafflesDict
...
constant
backAndCompGrid
system
constant
dynamicMeshDict
transportProperties
turbulentProperties
...
0

Calling the overset library: dynamicMeshDict

Open dynamicMeshDict:

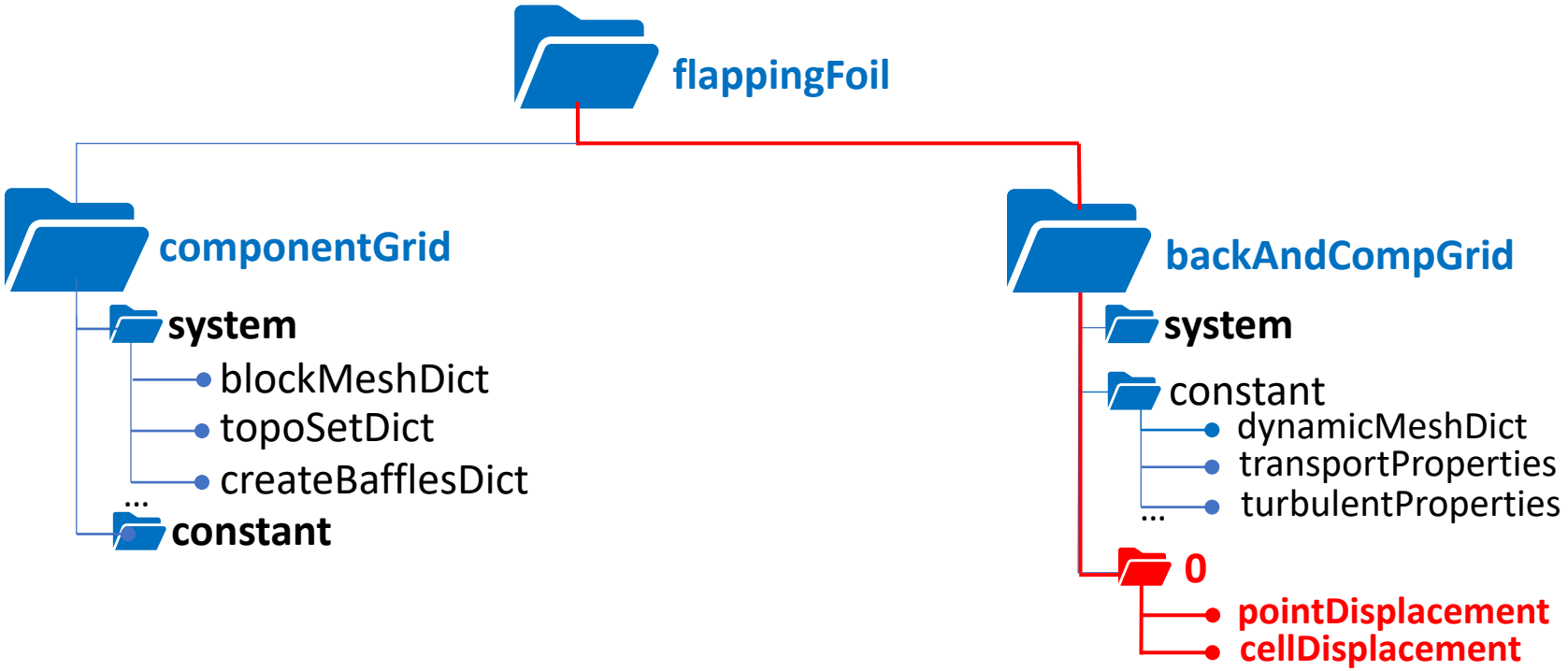
```
dynamicFvMesh       dynamicOversetFvMesh;

solver            displacementLaplacian;
cellZone        movingZone;

displacementLaplacianCoeffs
{
    diffusivity directional (10 70 0);
}
```

Mesh motion library: overset

At this stage those lines can be seen as place holders; only used when deformation comes in the picture (see later)

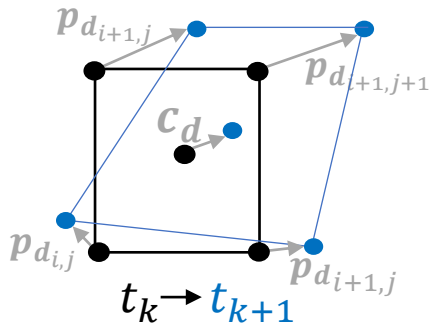
The next step is to impose the wing motion through the boundary conditions

Roadmap


flappingFoil
componentGrid
system
blockMeshDict
topoSetDict
createBafflesDict
...
constant
backAndCompGrid
system
constant
dynamicMeshDict
transportProperties
turbulentProperties
...
0
pointDisplacement
cellDisplacement

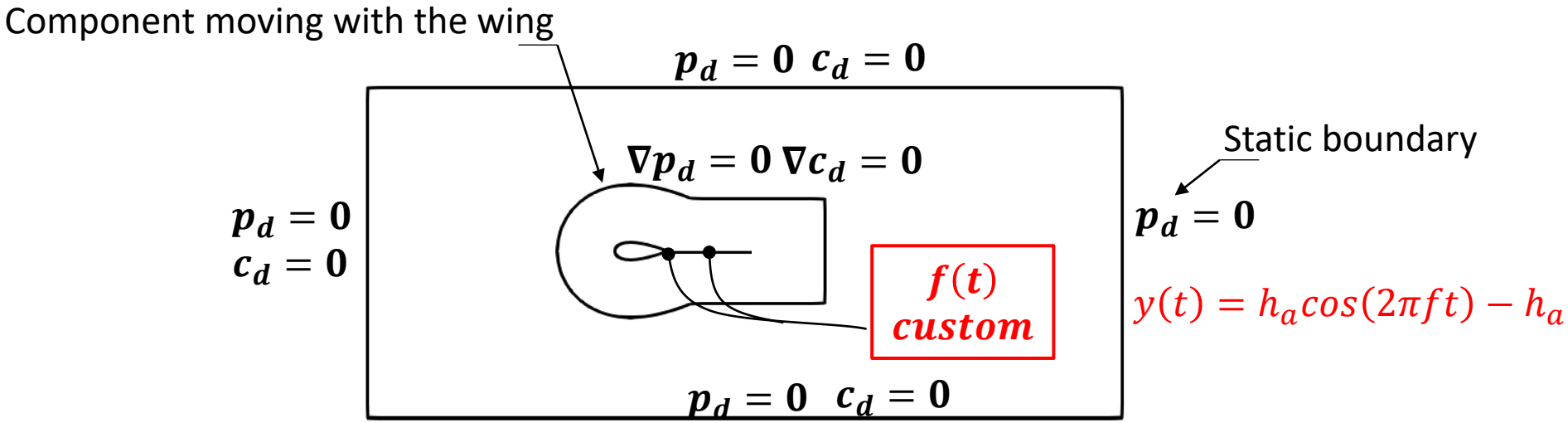
Boundary conditions

Point displacement p_d: displacement vector of the vertices of the mesh
Cell displacement p_d: displacement vector of the cell centres of the mesh



Those will define the wing motion*

In **0/pointDisplacement,** we need to impose the following:



The heaving motion we seek to simulate is not available in the default OpenFOAM library. **We have to implement it**

*other possibilities are avaible, see the OpenFOAM tutorial folder

Custom BC in OpenFOAM: preamble

- To save time*, we have already prepared the libraries for you. Copy the fvMotionSolver folder from the flappingFoil folder into your local src repository:

```
$ cp −r /your_case_path/flappingFoil/fvMotionSolver /your_src_path/
```

- Go into the repository where the custom BC must be edited:

```
$ cd /your_src_path/fvMotionSolver/pointPatchFields/derived/
```

- We will start by editing the heaving boundary condition:

```
$ cd heaving
```

*Complete procedure to edit a BC from an existing BC is available in appendix

Heaving BC: implementation

1. Open the header file and look at the declared variables
2. Use those variables to implement the heaving motion in the *updateCoeffs* function:

```
1  void Foam::heavingPointPatchVectorField::updateCoeffs()
2  {
3      if (this->updated())
4      {
5          return;
6      }
7
8      const polyMesh& mesh = this->internalField().mesh()();
9      const Time& t = mesh.time();
10
11     //const scalarField points( offset_ & patch().localPoints());
12
13     Field<vector>::operator=
14     (
15         ha_*cos(2*M_PI*frequency_*t.value()) - ha_
16     );
17
18     fixedValuePointPatchField<vector>::updateCoeffs();
19 }
```

$$y(t) = h_a cos(2\pi f t) - h_a$$

3. Make sure the C file is included in the Make folder
4. Compile:

```
$ wmake
```

Heaving BC: case setup

1. Go back to the test case directory
2. Open **system/controlDict** and include the new library at the beginning of the file:

```
libs            (overset fvMotionSolvers "mylibfvMotionSolvers.so");
```

3. Edit the pointDisplacement boundary conditions: use the heaving BC for the airfoil and the plate

```
    airFoil
    {
    patchType  overset;
    type       heaving;
    value uniform (0 0 0);

    ha         $ha;
    frequency $f;
    }
```

The parameters are defined in 0/include/initialConditions

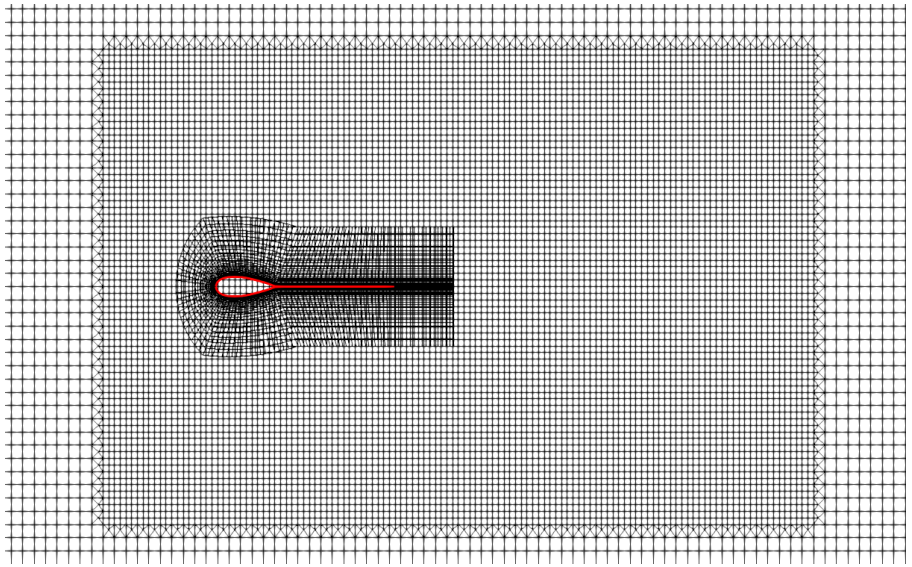
4. Run the mesh motion solver:

```
$ moveDynamicMesh –noFunctionObjects
```

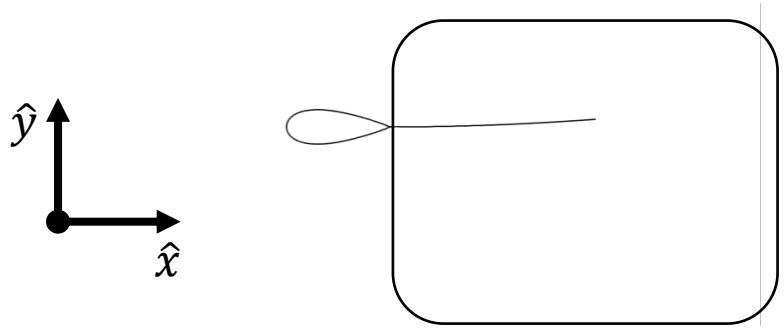
- Two main advantages:
  - The other BC field doesn't need to be defined
  - The time step can be made much larger than if the flow was also solved

Result

Heaving without deformation



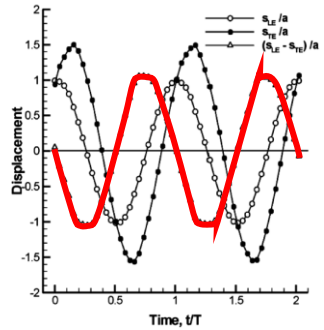
Heaving with deformation



- **No FSI simulation:** we impose the deformation of the plate according to [1]
- **Hypothesis:** the plate only deforms along $\hat{y}$ according to:

$$\boldsymbol{y(x,t) = h_p x^2 \sin(2\pi f t)}$$

where h_p has been calibrated to match the TE motion experienced in [1]:



$y(x_{TE}, t)$

Heaving and deforming BC

1. Go into the heavingAndSquareDeforming folder
2. Open the header file and look at the 2 new variables that have been declared
3. Use it to edit the ***updateCoeffs*** function:

```
scalarField xCoord = patch().localPoints().component(vector::X) - xoff_;

Field<vector>::operator=
(
    (ha_*cos(2*M_PI*frequency_*t.value()) - ha_)
+ hp_*sin(2*M_PI*frequency_*t.value())*sqr(xCoord)
);
```

$$y(t) = \underbrace{[h_a cos(2\pi f t) - h_a]}_{\text{heaving}} + \underbrace{h_p\left(x - x_{off}\right)^2 \sin(2\pi f t)}_{\text{deforming}}$$

4. Make sure the C file is included in the Make folder
5. Compile:

```
$ wmake
```

6. Go to the test case directory and edit the BC of the plate (master and slave)

DynamicMeshDict

Until now, we have ignored part of the **dynamicMeshDict** that contains the library for mesh deformation.
Open constant/dynamicMeshDict:

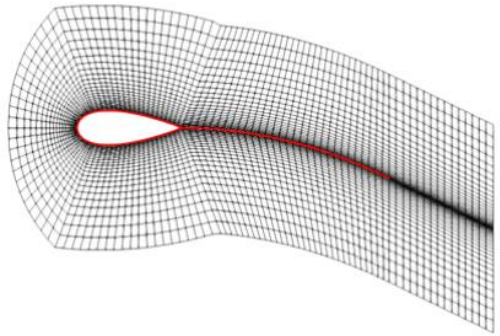
```
dynamicFvMesh       dynamicOversetFvMesh;

solver            displacementLaplacian;
cellZone          movingZone;
displacementLaplacianCoeffs
{
    diffusivity directional (10 70 0);
}
```

The movingZone is using the Laplace equation for $\boldsymbol{c_d}$ to deform its cells according to the plate motion

Laplace smoothing

$$\nabla \cdot (\boldsymbol{\gamma} \nabla \boldsymbol{c_d}) = 0$$

$$\boldsymbol{c_{k+1}} = \boldsymbol{c_k} + \boldsymbol{c_d}$$



Set the diffusivity $\boldsymbol{\gamma}$ to conserve high quality cells near the plate

- **Option 1:** $\boldsymbol{\gamma} = \boldsymbol{cste}$
- **Option 2:** $\boldsymbol{\gamma} \backsim (\frac{1}{l^x}, \frac{1}{l^x}, \frac{1}{l^x})$
- **Option 3:** $\boldsymbol{\gamma} \backsim (a, b, c)$
- **Other options***

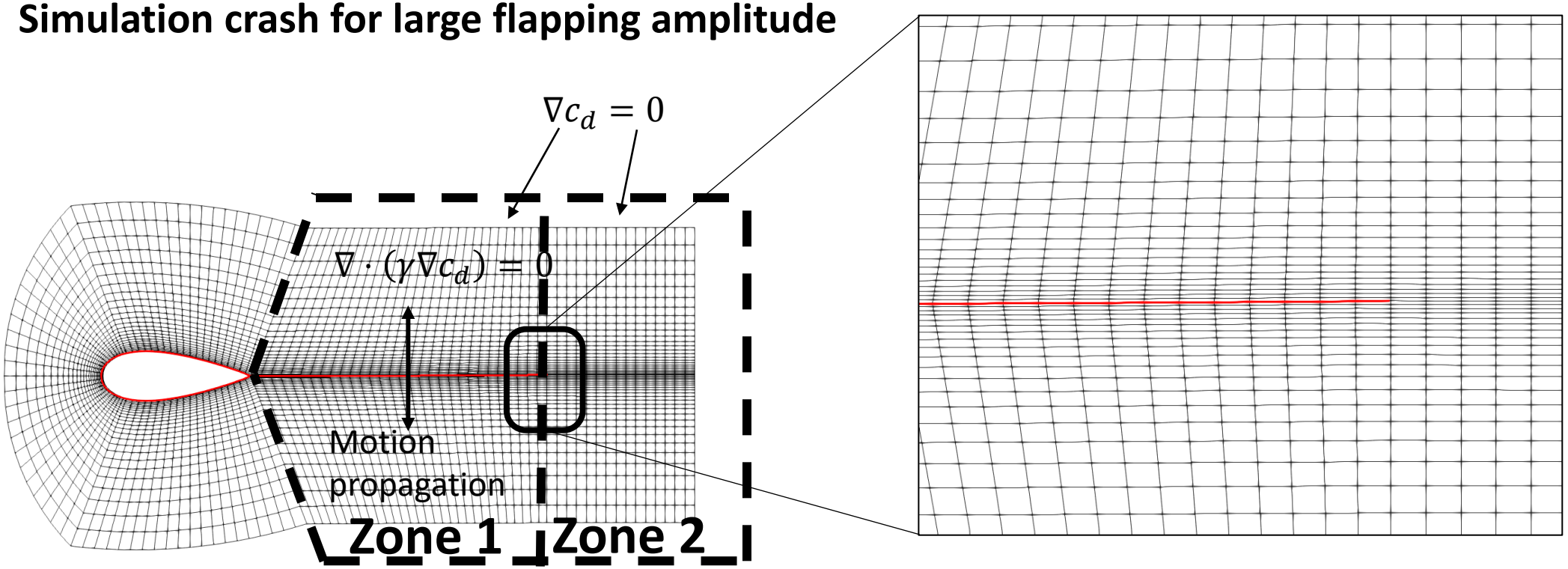
*More info? https://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2022/AndreDaLuzMoreira/Report_AndreDLM.pdf

Morping mesh issue

Remove saved time steps (if present) and run the mesh motion solver:

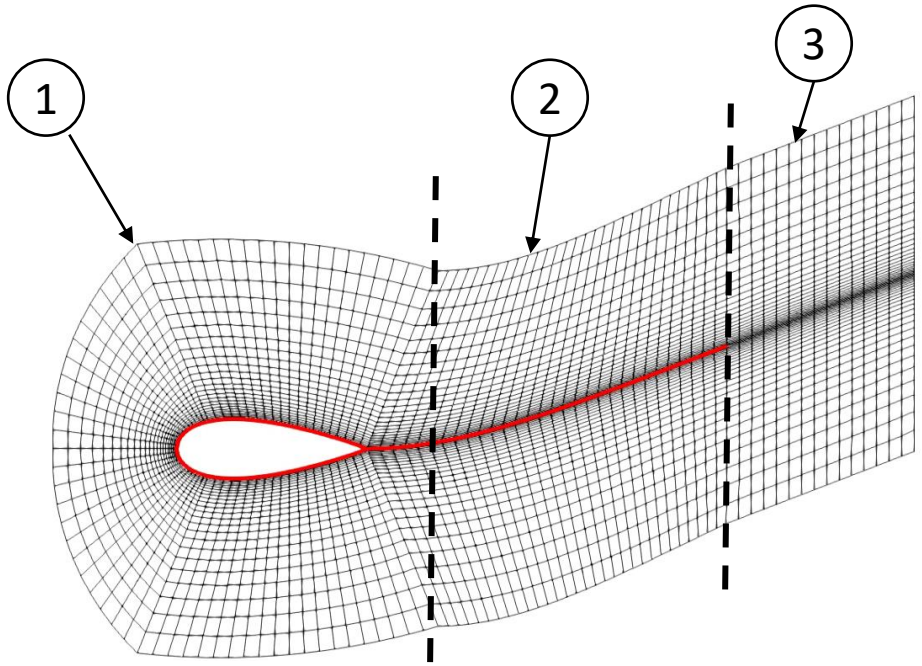
```
$ moveDynamicMesh –noFunctionObjects
```

× **Simulation crash for large flapping amplitude**



We will impose a custom BC on the component grid to circumvent this issue

Component grid BC



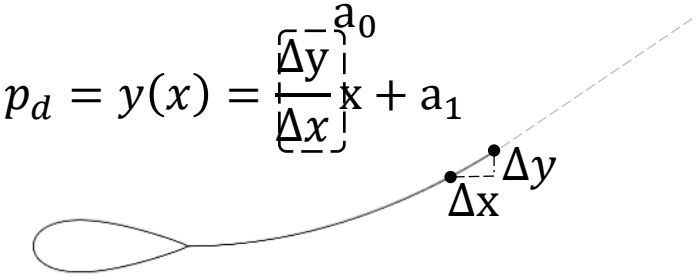
1	$h_a cos(2\pi f t) - \mathrm{h_a}$	$x < c + tol$
2	$h_a cos(2\pi f t) - \mathrm{h_a} + y_0 x^2 \sin(2\pi f t)$	$c + tol < x < c + l$
3	$h_a cos(2\pi f t) - \mathrm{h_a} + (\mathrm{a_0 x + a_1})\sin(2\pi f t)$	$x > c + l$

Replace the BC of the cellDisplacement field to:

```
1     oversetPatch
2     {
3         patchType       overset;
4         type            cellMotion;
5         value           uniform (0 0 0);
6     }
```

$$p_d = y(x) = \frac{\Delta y}{\Delta x} x + \mathrm{a_1}$$

$$\mathrm{a_0} = \frac{\Delta y}{\Delta x}$$



Component grid BC

1. Go into the heavingAndLinearDeforming folder
2. Open the header file and look at the 3 new variables that have been declared
3. Implement the heaving and deforming function in the *updateCoeffs:*

```
1    scalarField xCoord = patch().localPoints().component(vector::X);
2    scalarField flag1 = xCoord;
3    scalarField flag2 = xCoord;
4
5    forAll(xCoord, faceI)
6    {
7    flag1[faceI] = (xCoord[faceI] ⩽  (xoff_+0.01346)) ? 1.0 : 0.0;
8    flag2[faceI] = (xCoord[faceI] ⩾ (xoff_ +length_)) ? 1.0 : 0.0;
9    }
10
11   Field<vector>::operator=
12   (
13       (ha_*cos(2*M_PI*frequency_*t.value()) - ha_) +
14   (1-flag1)*(1-flag2)*hp_*sin(2*M_PI*frequency_*t.value())*sqr(xCoord-xoff_)
15       + flag2*(a0_*(xCoord)+a1_)*sin(2*M_PI*frequency_*t.value())
16   );
```

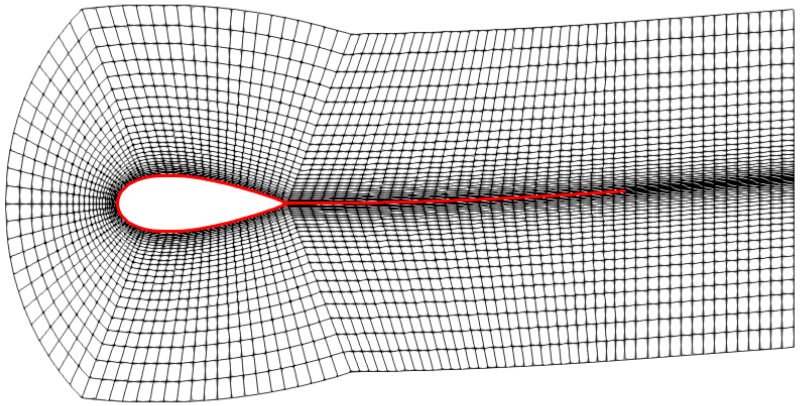
4. Compile

```
$ wmake
```

5. Go to the test case directory, edit the $\boldsymbol{p_d}$ boundary condition of the oversetPatch, remove saved time steps and run:

```
$ moveDynamicMesh –noFunctionObjects
```

Results



(Camera following the heaving motion)

We are only one step away from running the simulation ☺

Case set-up

- The test case is:
  - Unsteady
  - Incompressible
  - Dynamic mesh

  } **overPimpleDyMFoam** with adaptative time steps

- The flow is assumed ~laminar ($Re = 9k$) and the fluid is water. Open constant/turbulenceProperties and constant/transportProperties

```
simulationType laminar;
```

```
transportModel  Newtonian;

nu              nu [ 0 2 -1 0 0 0 0 ] 1e-06;
rho             rho [ 1 -3 0 0 0 0 0 ] 1000;
```

- Open fvSolution and check the solver settings

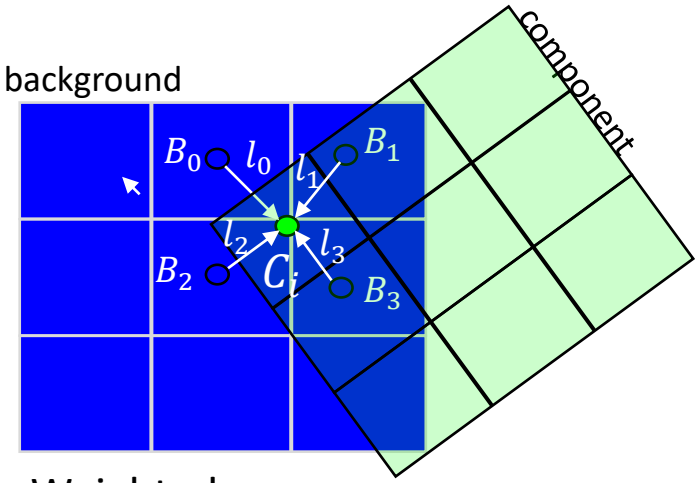
```
PIMPLE
{
    momentumPredictor   true;
    correctPhi          false;
    oversetAdjustPhi    false;
    nOuterCorrectors    1;
    nCorrectors         4;
    nNonOrthogonalCorrectors 0;

    ddtCorr             true;

    pRefPoint          1000;
    pRefValue         0;
}
```

By default, overset interpolations are not conservative. oversetAdjustPhi enforce mass conservation by correcting the fluxes of each cell

Case set-up: fvSchemes

Open fvSchemes:

```
ddtSchemes
{
    default         Euler;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div(phid,p)      Gauss linearUpwind grad(p);
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
    div((nuEff*dev2(T(grad(U))))   Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
(...)
oversetInterpolation
{
   method          inverseDistance;
}
(...)
```



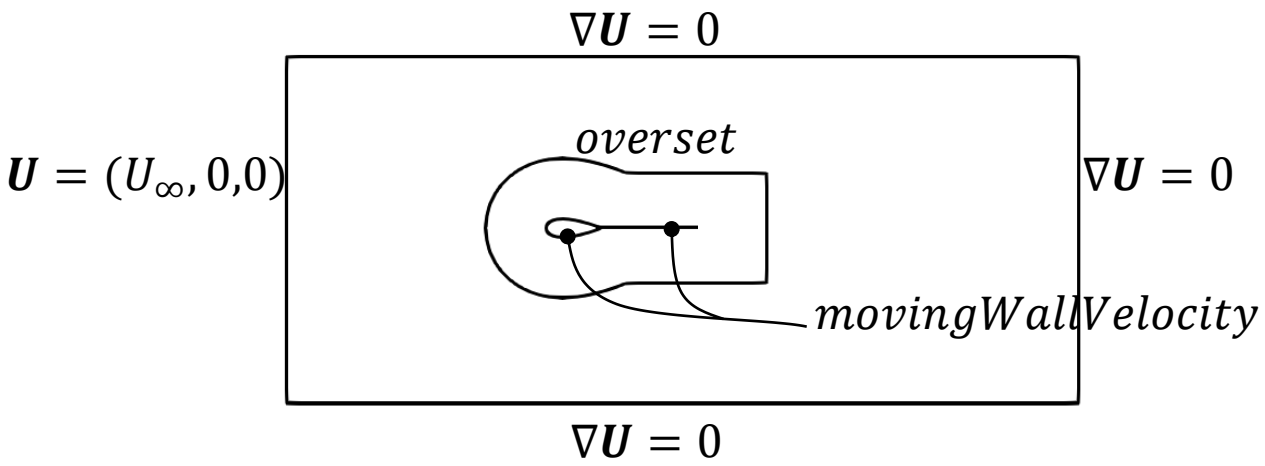
Weighted average

$$\phi_{C_i} = \sum_{k=1}^{N_D} w_k \phi_{B_k}$$

where $w_k = 1/l_k$

Case set-up: boundary conditions

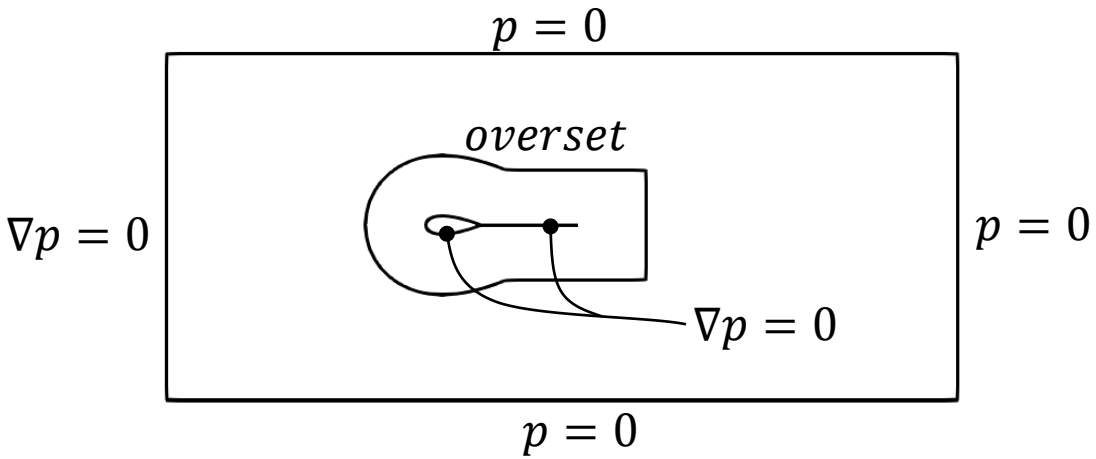
Finally open 0/U




```
oversetPatch
    {
        type            overset;
    }
 inlet
    {
        type            fixedValue;
        value           uniform $flowVelocity;
    }
    "(outlet|upperPatch|lowerPatch)"
    {
        type            zeroGradient;

    }
    "(airFoil|flatPlate_master|flatPlate_slave)"
    {
        type            movingWallVelocity;
        value uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }
```

Case set-up: boundary conditions

Finally open 0/p …




```
oversetPatch
{
    type            overset;
}
outlet
{
    type            fixedValue;
    value           uniform $pressure;
}
"(inlet|upperPatch|lowerPatch)"
{
    type            zeroGradient;
}
"(airFoil|flatPlate_master|flatPlate_slave)"
{
    type            zeroGradient;
}
frontAndBack
{
    type            empty;
}

```

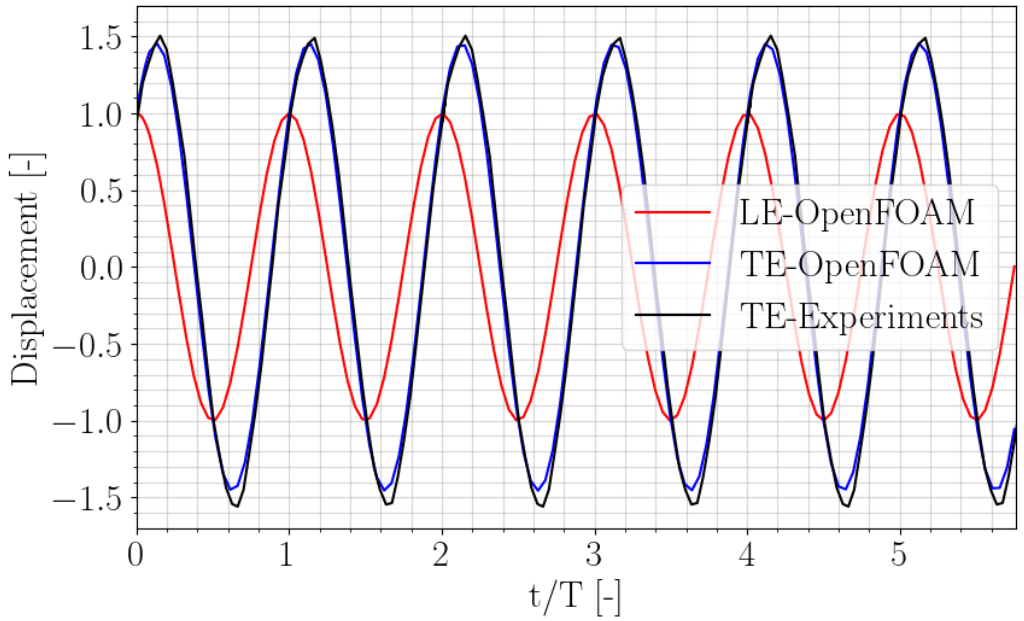
… and run the simulation:

```
$ overPimpleDyMFoam
```

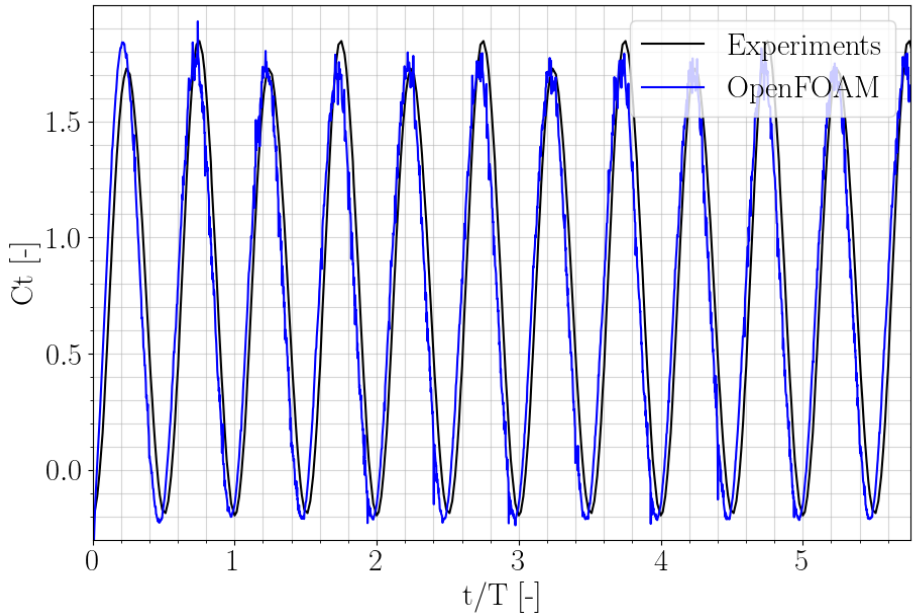
Results

Validation

Plate displacement

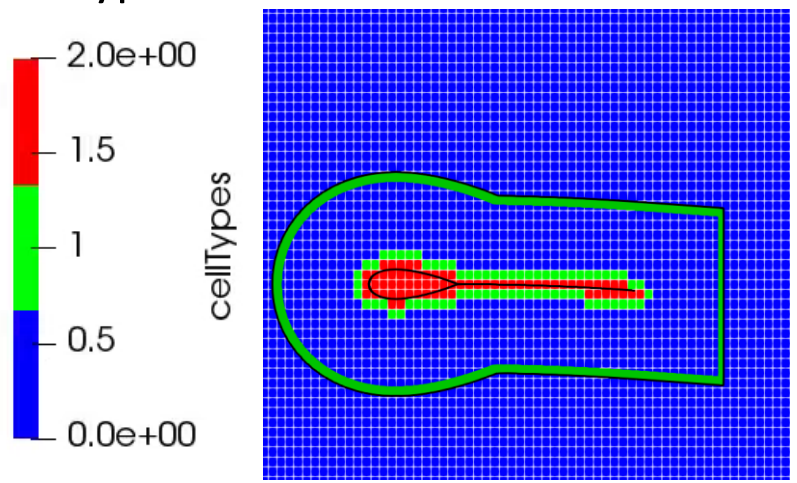


Thrust



Post-process

1. Open Paraview
2. Tick the “With sets” box
3. Tick only “component” and “background” cellSet boxes in the “Mesh Parts”
4. Tick cellTypes in the “Volume Fields” section
5. Press apply
6. Select the “Extract Block” filter and tick only the component grid
7. Press apply and decrease the opacity of the “Extract Block” filter
8. Make visible both items in the “Pipeline Browser” and display the cellTypes
9. Press play



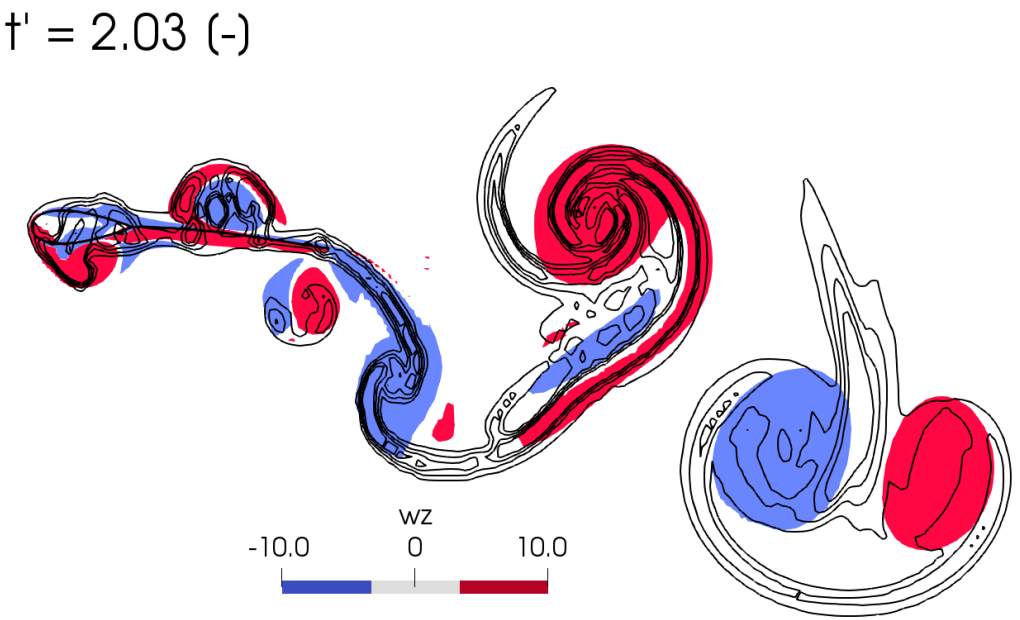
Post-process

1. Generate the vorticity field:

```
$ postProcess −func vorticity
```

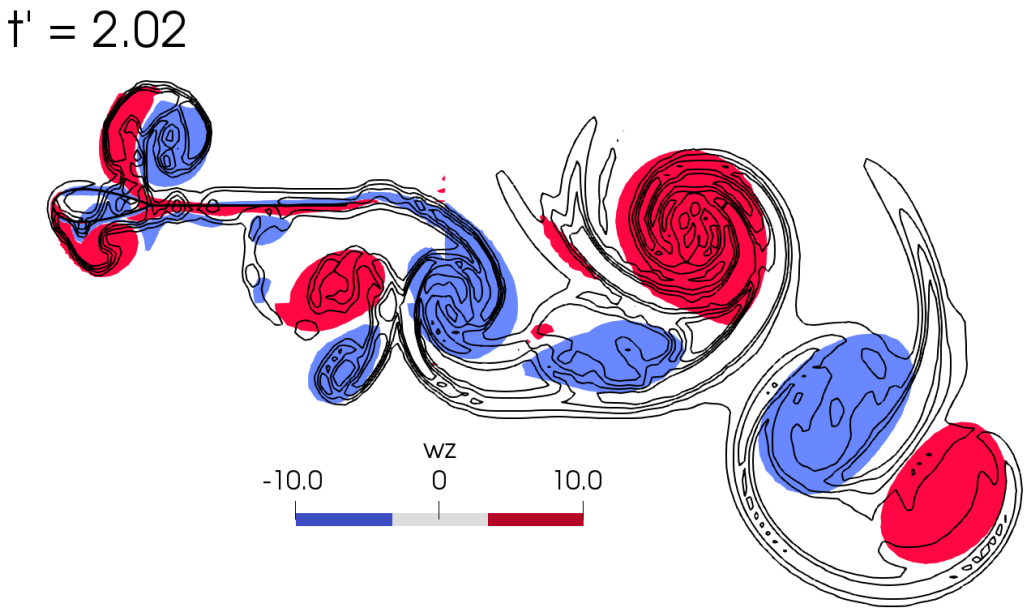
2. Show the z-component of the vorticity field for both items in the “Pipeline Browser”

Can you see why the wing generate a mean thrust and not a mean drag?

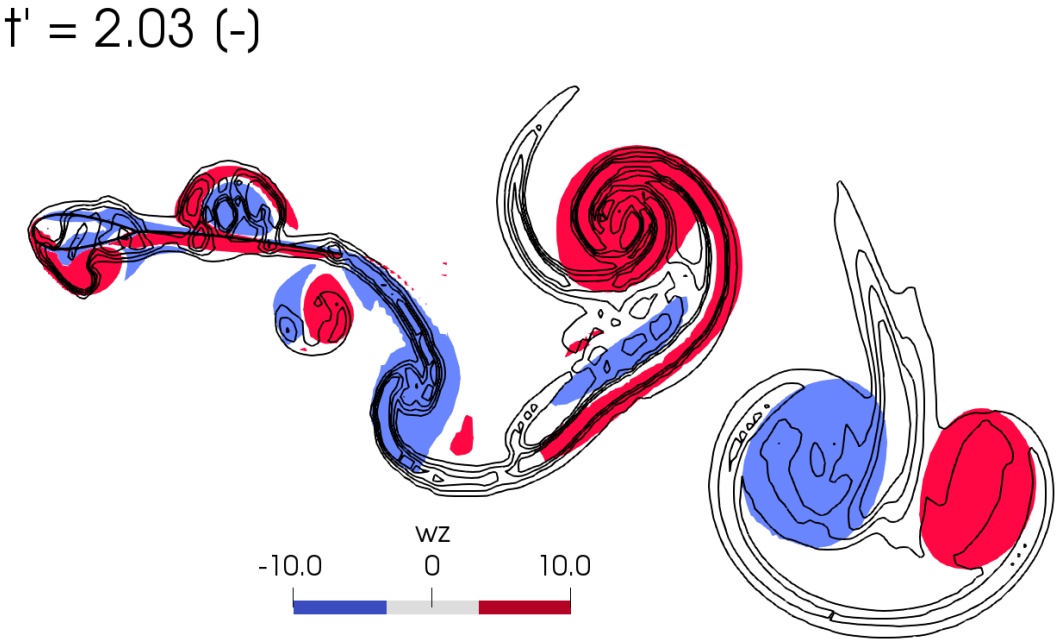


Influence of flexibility

Stiff



Flexible



Analysis in:

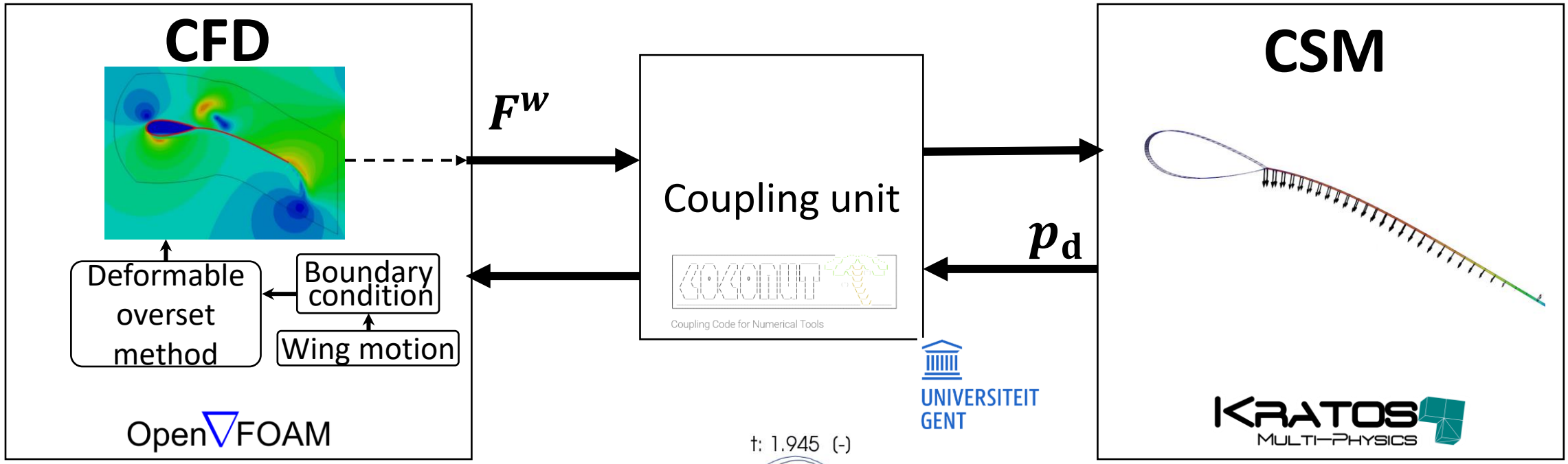
Poletti, R., Barucca, M., Koloszar, L., Mendez, M., & Degroote, J. (2023). Development of an FSI environment for the aerodynamic performance assessment of flapping wings. In *X International Conference on Computational Methods for Coupled Problems in Science and Engineering*.

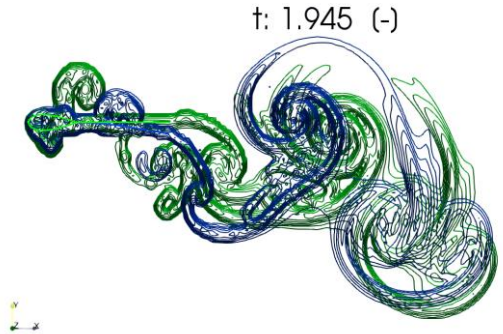
Conclusion

- In this tutorial, we have:
  - Gone through all the steps needed to defined an overset test case that uses a baffle
  - Couple the overset method with the morphing mesh technique
  - Circumvent its limit through the implementation of custom boundary conditions

What would you like to work on for a next VKI OF seminar ?

For a next VKI OF seminar?





Coconut is an open-source coupling code developed at Ghent University

Want to try out an FSI simulation? https://pyfsi.github.io/coconut/

References

[1] Heathcote, S., & Gursul, I. (2007). Flexible flapping airfoil propulsion at low Reynolds numbers. *AIAA journal*, *45*(5), 1066-1079.

[2] Hadzic, H. (2006). *Development and application of finite volume method for the computation of flows around moving bodies on unstructured, overlapping grids*. Technische Universität Hamburg

[3] https://openfoamwiki.net/index.php/OverPimpleDyMFoam

[4]https://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2022/AndreDaLuzMoreira/Report_AndreDLM.pdf

[5] Tisovska, P. (2019). Description of the overset mesh approach in ESI version of OpenFOAM. *Proceedings of CFD with OpenSource Software*.

[6] Poletti, R., Barucca, M., Koloszar, L., Mendez, M., & Degroote, J. (2023). Development of an FSI environment for the aerodynamic performance assessment of flapping wings. In *X International Conference on Computational Methods for Coupled Problems in Science and Engineering*.

Thank you for your attention

Custom BC in OpenFOAM: preamble (1/2)

1. **Create a local source code repository** that will contain your custom boundary conditions. The path of this directory is up to you

```
$ mkdir src
```

2. **Copy the fvMotionSolver folder** of OpenFOAM into your local src. It contains the boundary conditions of the $\boldsymbol{pD}$ field:

```
$ cp –r $FOAM_SRC/fvMotionSolver /your_src_path/
```

3. Go into the fvMotionSolver/pointPatchFields/derived and **create a copy of the waveDisplacement** BC:

```
$ cd /your_ path /fvMotionSolver/pointPatchFields/derived
$ cp –r waveDisplacement heaving
```

4. Go into the heaving folder and **rename the source and header files**:

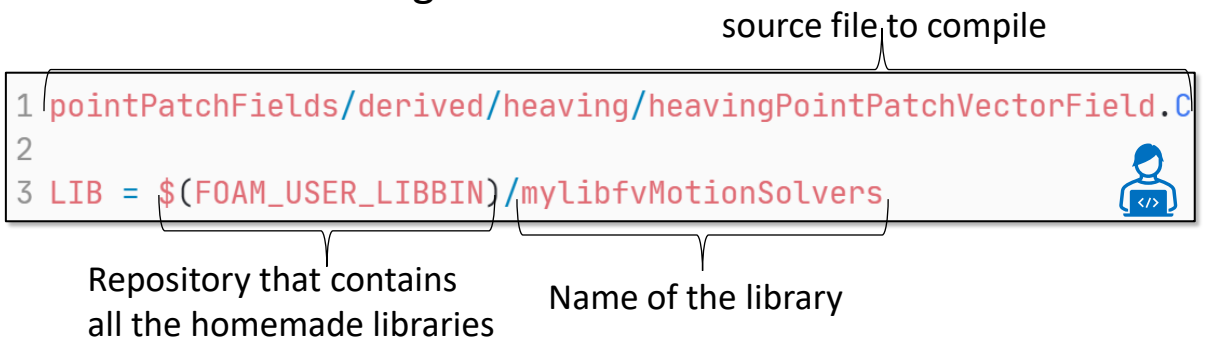
```
$ cd heaving
$ find . -name ‘wave.*’ –exec rename waveDisplacement heaving {} \;
```

5. Rename all the occurrences of the name waveDisplacement inside both files:

```
$ sed -i 's\waveDisplacement\heaving\g' heavingPointPatchVectorField.*
```

Custom BC in OpenFOAM: preamble (2/2)

6. Go three folders below. There should be the **Make folder** which contains two files (at least):
   - ***options:*** list of libraries taken from the original compilation of OpenFOAM
   - ***files:***
     - Contains the names of the C files to compile and the name of the resulting library
     - Open and edit it with a similar setting as:



   - **Compile**; normally no errors should appear as you have not yet edited the boundary condition

```
$ wmake
```

Let's start by defining a BC that imposes a heaving motion