Online Training – Advanced session
July 2021

Dynamic meshes in OpenFOAM:

Mesh morphing, overset meshes, sliding meshes, moving bodies, rigid body motion, and adaptive mesh refinement

wolf dynamics

Copyright and disclaimer

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

© 2014-2021 Wolf Dynamics.

All rights reserved. Unauthorized use, distribution or duplication is prohibited.

Contains proprietary and confidential information of Wolf Dynamics.

Wolf Dynamics 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 Wolf Dynamics 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.

All trademarks are property of their owners.

Revision 2-2021
JG

Before we begin

On the training material

- **This training is based on OpenFOAM 8 and OpenFOAM 2012 or newer (for overset meshes).**
- In the USB key/downloaded files you will find all the training material (tutorials, slides, and lectures notes).
- You can extract the training material wherever you want. From now on, this directory will become:
  - **`$TM`**
    (abbreviation of **T**raining **M**aterial)
- To uncompress the tutorials go to the directory where you copied the training material (**`$TM`**) and then type in the terminal,
  - `$> tar -zxvf file_name.tar.gz`
- In every single tutorial, you will find the file `README.FIRST`. In this file you will find the general instructions of how to run the case. You will also find some additional comments.
- In some cases, you will also find additional files with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, `run_plot.sh`, and so on.
- These files can be used to run the case automatically by typing in the terminal, for example,
  - `$> sh run_solver.`
- However, if you are not familiar with OpenFOAM, we highly recommend that you open the `README.FIRST` file and then type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Conventions used

- The following typographical conventions are used in this training material:

- Text in `Courier new` font indicates Linux commands that should be typed literally by the user in the terminal.
- Text in **`Courier new bold`** font indicates directories.
- Text in *`Courier new italic`* font indicates human readable files or ascii files.
- Text in **Arial bold font** indicates program elements such as variables, function names, classes, statements and so on. It also indicates environment variables, and keywords. They also highlight important information.
- Text in Arial underline in blue font indicates URLs and email addresses.
- This icon indicates a warning or a caution.
- This icon indicates a tip, suggestion, or a general note.
- This icon indicates a folder or directory.
- This icon indicates a human readable file (ascii file).
- This icon indicates that the figure is an animation (animated gif).
- These characters `$>` indicate that a Linux command should be typed literally by the user in the terminal.

Conventions used

- The following typographical conventions are used in this training material:

  - Large code listing, ascii files listing, and screen outputs can be written in a square box, as follows:

```
1    #include <iostream>
2    using namespace std;
3
4    // main() is where program execution begins.  It is the main function.
5    // Every program in c++ must have this main function declared
6
7    int main ()
8    {
9            cout << "Hello world";                    //prints Hello world
10           return 0;                                 //returns nothing
11   }
```

  - To improve readability, the text might be colored.
  - The font can be `Courier new` or **Arial bold**.
  - And when required, the line number will be shown.

Roadmap

1. Introduction – What are dynamic meshes?
2. Adaptive mesh refinement in OpenFOAM
3. Sliding meshes in OpenFOAM
4. Morphing meshes in OpenFOAM
5. Moving meshes in OpenFOAM
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

Roadmap

1. **Introduction – What are dynamic meshes?**
2. Adaptive mesh refinement in OpenFOAM
3. Sliding meshes in OpenFOAM
4. Morphing meshes in OpenFOAM
5. Moving meshes in OpenFOAM
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

Introduction – What are dynamic meshes?

What are dynamic meshes?

- Dynamic meshes are models that allows the mesh to change during a simulation.
- The changes of the mesh can be due to a prescribed motion, rigid body motion, fluid structure interaction or refinement/unrefinement.
- Dynamic meshes are compatible with all physical models (as useual, there are a few exceptions).
- Dynamic meshes can be used to:
  - Morph the mesh to accommodate the body motion.
  - Add remove new cells according to a criterion.
  - Add/remove layers (layering).
  - Sliding meshes.
  - Interpolate the solution at matching patches.
  - Compute the solution in overset meshes.
  - Deform bodies.
  - Fluid structure interaction (FSI).

Introduction – What are dynamic meshes?

What are dynamic meshes?

- In dynamic meshes (specifically when working with moving/deforming bodies), the integral form of the general transport equation is written as follows,

$$\int_{V_p} \frac{\partial \rho \phi}{\partial t} dV + \int_{V_p} \nabla \cdot [\rho \phi (\mathbf{u} - \mathbf{u}_g)] dV - \int_{V_p} \nabla \cdot (\rho \Gamma_\phi \nabla \phi) dV = \int_{V_p} S_\phi \phi dV$$

Mesh velocity

- Also, the boundary condition of the moving/deforming walls takes into account the mesh velocity.
- In the FVM, we want to solve the general transport equation for the transported quantity ϕ in a given domain, with given boundary conditions BC and initial conditions IC.
- As you can see, the only difference with the standard general transport equation is the addition of the mesh velocity.
- The rest of the FVM formulation remains the same, except for a few considerations related to the moving mesh, as we will see.

NOTE: remember, starting from this equation we can write down the Navier-Stokes equations (NSE).

Introduction – What are dynamic meshes?

What are dynamic meshes?

- In dynamic meshes (specifically when working with moving/deforming bodies), the integral form of the general transport equation is written as follows,

$$\int_{V_p} \frac{\partial \rho\phi}{\partial t} dV + \int_{V_p} \nabla \cdot [\rho\phi(\mathbf{u} - \mathbf{u}_g)] dV - \int_{V_p} \nabla \cdot (\rho\Gamma_\phi \nabla\phi) dV = \int_{V_p} S_\phi \phi dV$$

Function of time

Mesh velocity

- Notice that the volume dV is a function of time.
- This implies that the shape of the cells is changing in time, and this might introduce mesh quality problems.

NOTE: remember, starting from this equation we can write down the Navier-Stokes equations (NSE).

What are dynamic meshes?

- The time derivative can be written as follows,

$$\frac{\partial}{\partial t}\int_{V_p}\rho\phi dV = \frac{(\rho\phi V)^{n+1} - (\rho\phi V)^{n}}{\Delta t}$$

Hereafter we are using first order backward difference formulation

- Where *n* and *n+1* denote the respective quantity at the current and next time level.
- Therefore, it is important to choose a time-step that does not result in large volume changes between iterations.

What are dynamic meshes?

- The (n+1) time level volume V^{n+1} can be computed as follows (volume update),

$$V^{n+1} = V^n + \frac{dV}{dt}\Delta t$$

Where dV/dt is the volume time derivative of the control volume

- In order to satisfy the grid conservation law, the volume time derivative dV/dt of the control volume can be computed as follows,

$$\frac{dV}{dt} = \int_{\partial V_p} \mathbf{u}_g \cdot d\mathbf{S} = \sum_f \mathbf{u}_{g_f} \cdot \mathbf{S}_f$$

$d\mathbf{S} = \mathbf{n}dS$

Face area vector

Summation among all faces *f* that make up the cell

Where the integrant has been approximated by means of the mid point rule

What are dynamic meshes?

- In order to satisfy the grid conservation law, the volume time derivative dV/dt of the control volume can be computed as follows,

$$\frac{dV}{dt} = \int_{\partial V_p} \mathbf{u}_g \cdot d\mathbf{S} = \sum_f \mathbf{u}_{g_f} \cdot \mathbf{S}_f$$

Where the integrant has been approximated by means of the mid point rule

$d\mathbf{S} = \mathbf{n} dS$

Face area vector

Summation among all faces *f* that make up the cell

- The dot product $\mathbf{u}_{g_f} \cdot \mathbf{S}_f$ on each control volume face is calculated as,

$$\mathbf{u}_{g_f} \cdot \mathbf{S}_f = \frac{\delta V_f}{\Delta t}$$

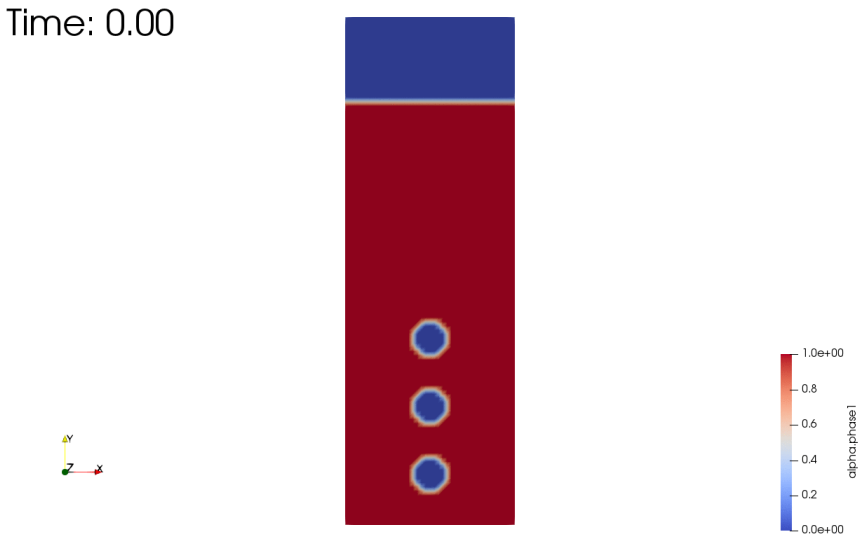
Volume swept out by the control volume face f over the time step Δt

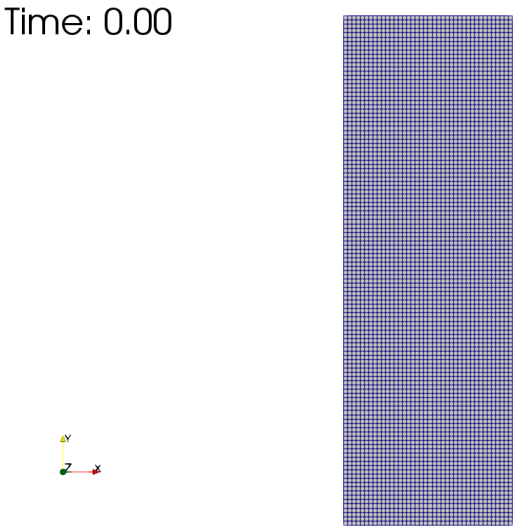
- To avoid stability and accuracy problems due to cell quality (as the cells are changing in time), the time step should be chosen in such a way that it does not result in large volume changes between iterations.

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Adaptive mesh refinement (AMR) – Three rising bubbles (VOF)





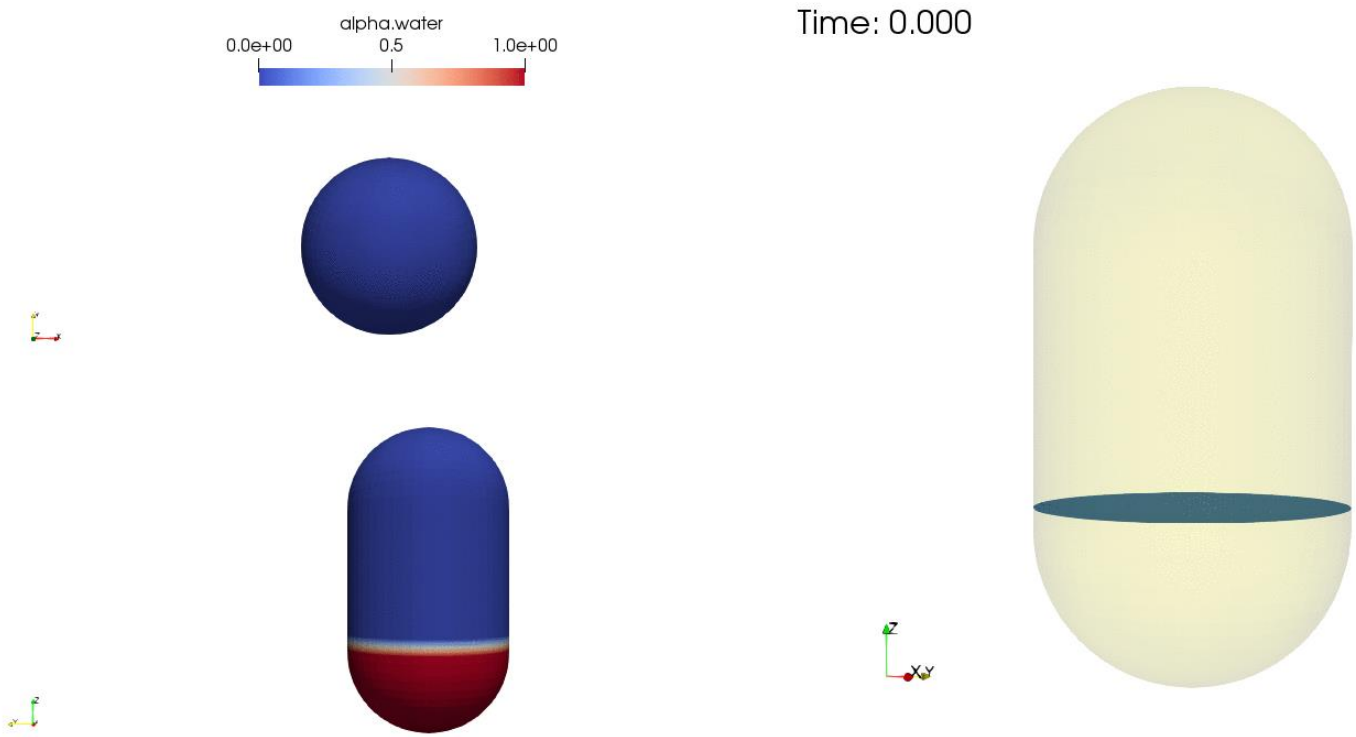
http://www.wolfdynamics.com/training/mphase/image2.gif

http://www.wolfdynamics.com/training/mphase/image3.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Moving domain – Sloshing tank

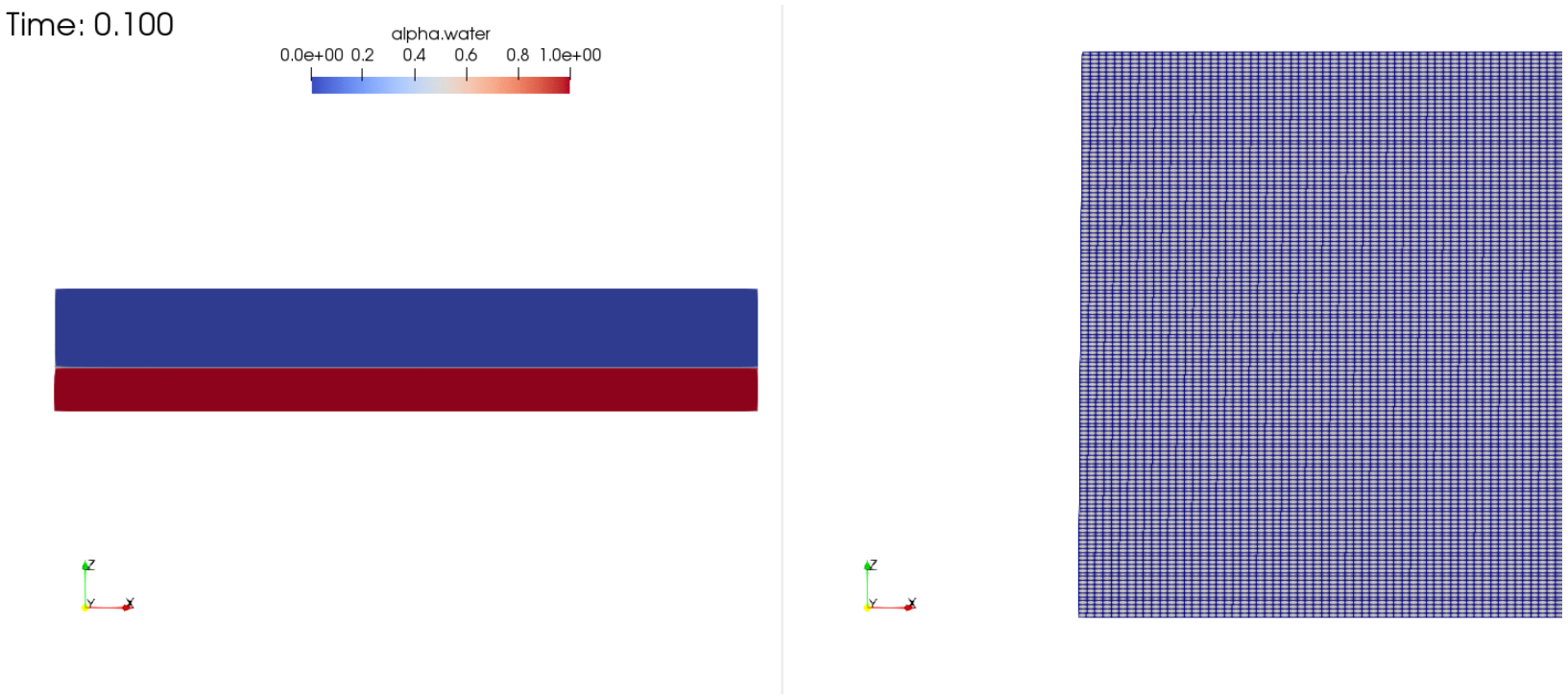


http://www.wolfdynamics.com/training/dynamicMeshes/sloshingCylinder.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Moving boundaries with mesh morphing – Wave maker (VOF)

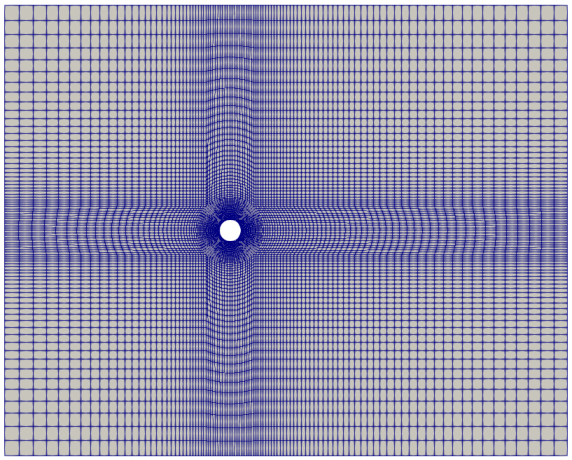


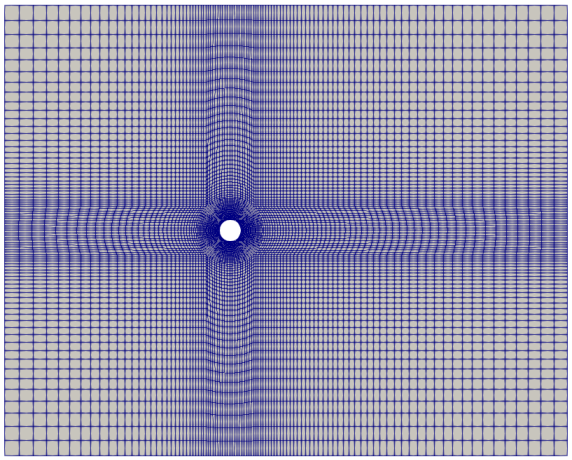
http://www.wolfdynamics.com/training/dynamicMeshes/waveMaker.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Mesh morphing – Different mesh smoothing methods





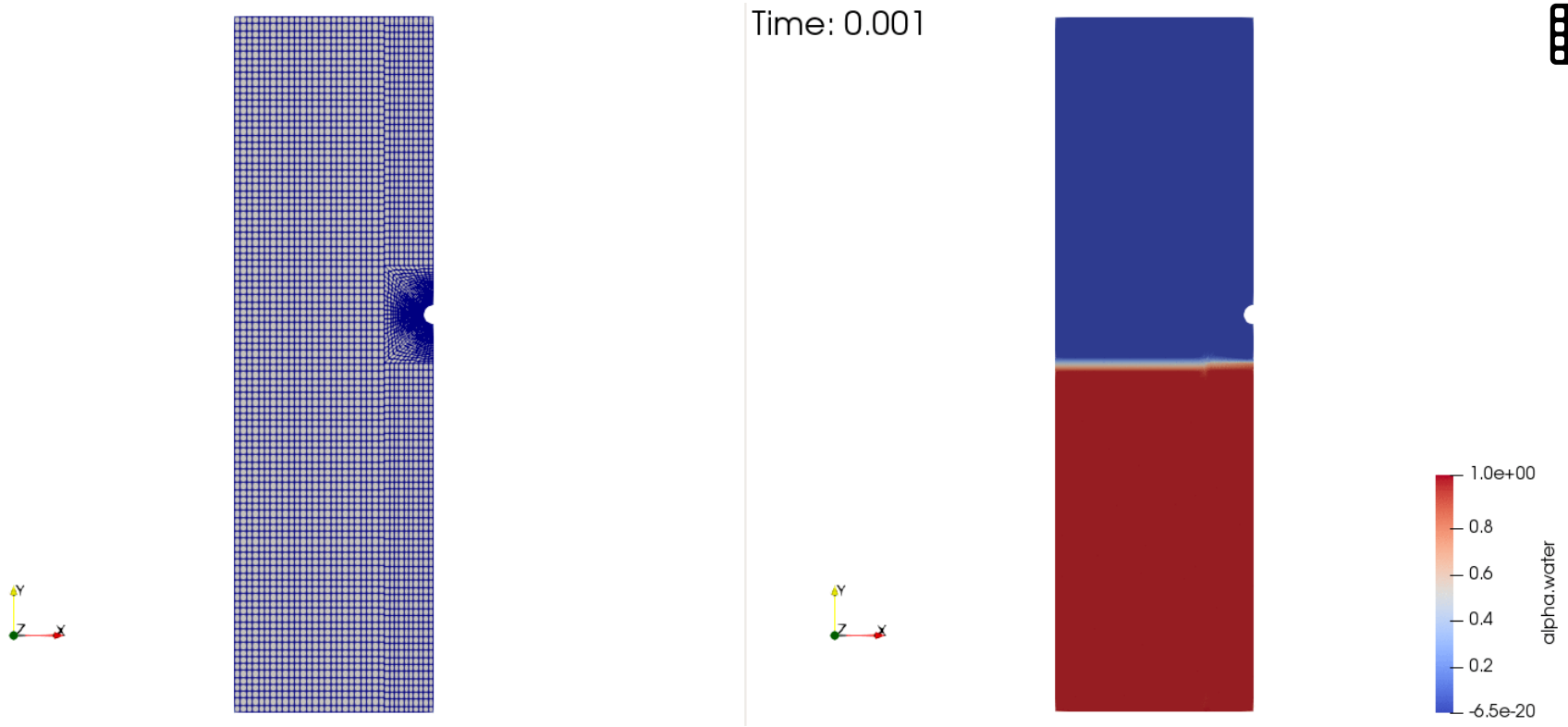
http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion1

http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion2

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Layering with mesh zones interface – Water impact

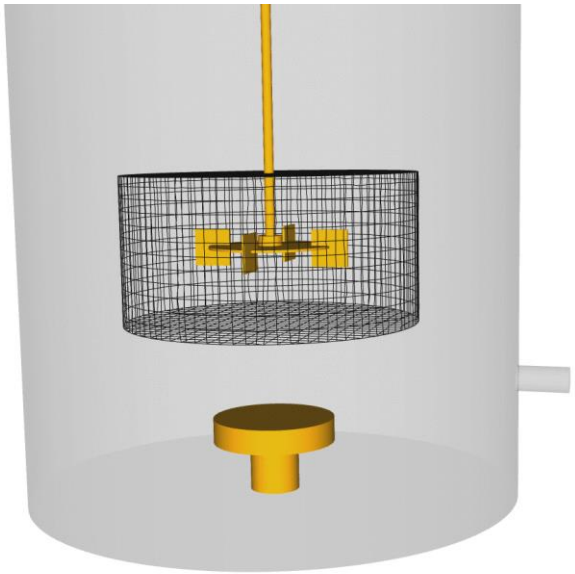


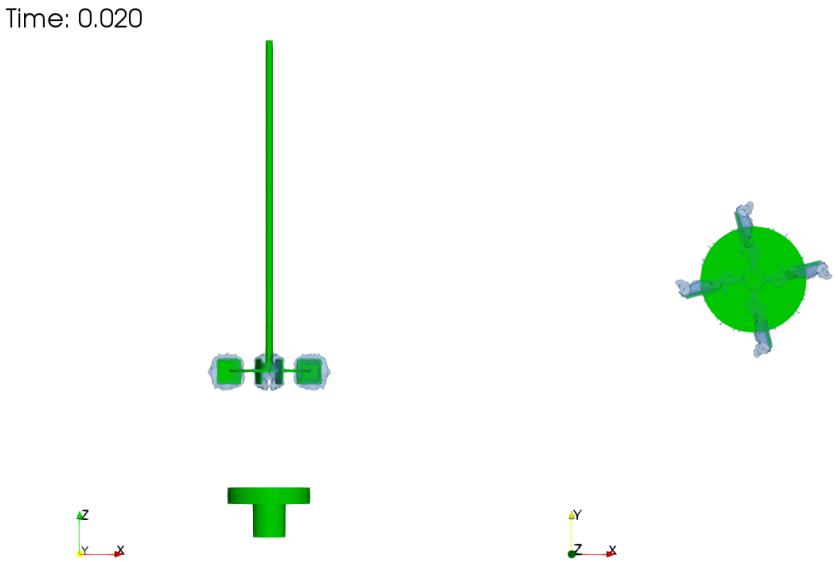
http://www.wolfdynamics.com/training/dynamicMeshes/layeringMesh.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Sliding meshes – Continuous stirring tank reactor (CSTR)





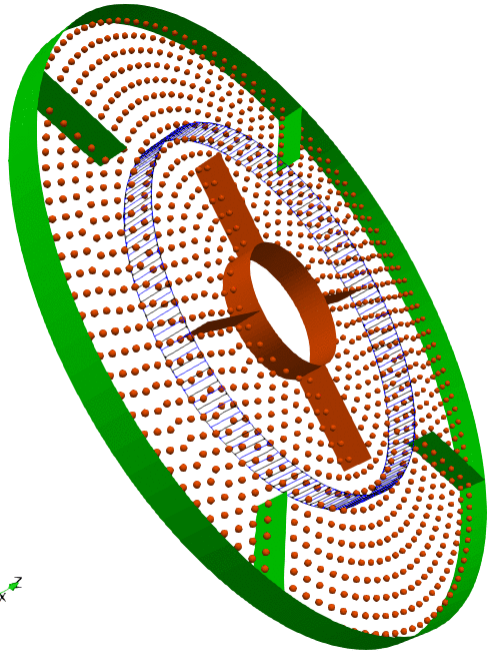
www.wolfdynamics.com/training/mphase/image8.gif

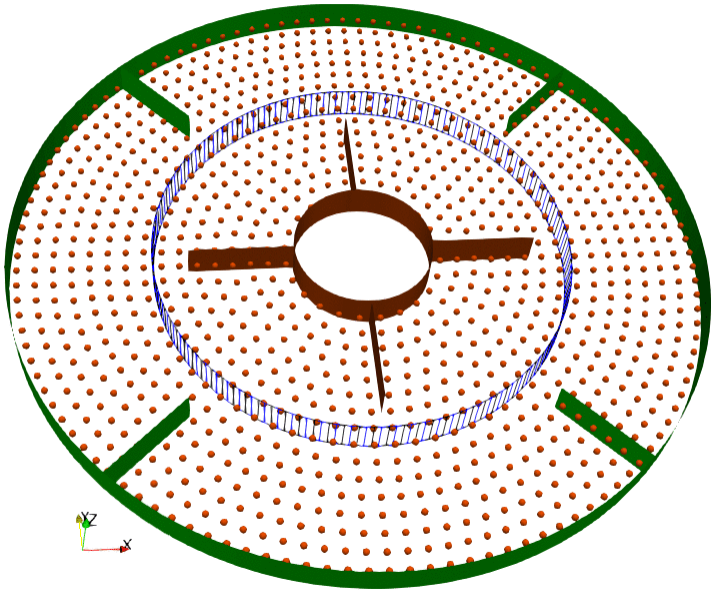
http://www.wolfdynamics.com/training/movingbodies/image13.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Sliding meshes with particles interaction



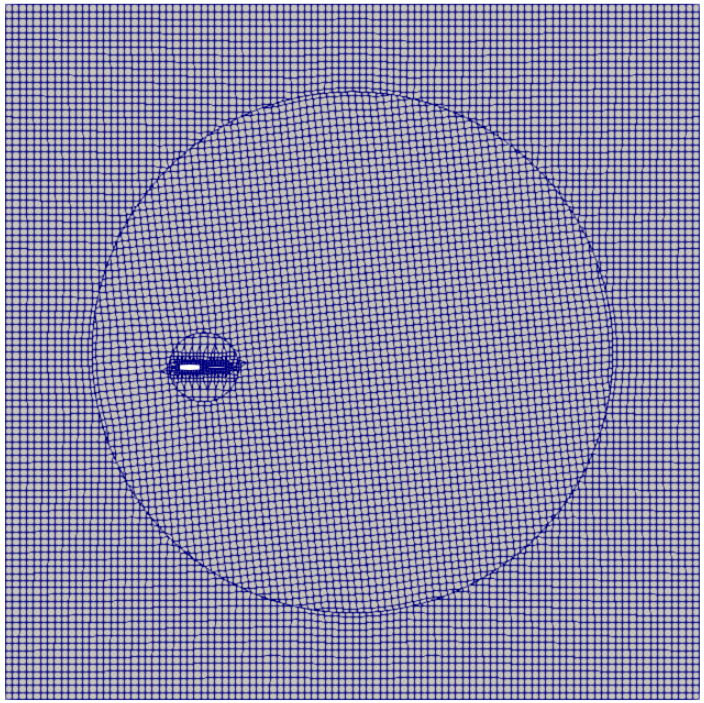


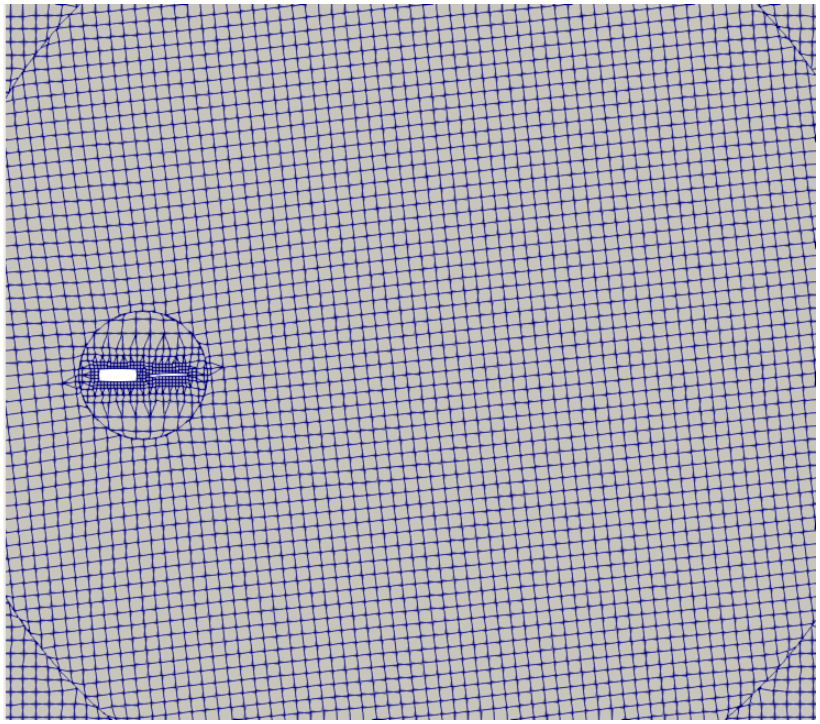
http://www.wolfdynamics.com/training/dynamicMeshes/sliding_particles.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Sliding meshes with relative motion and mesh morphing



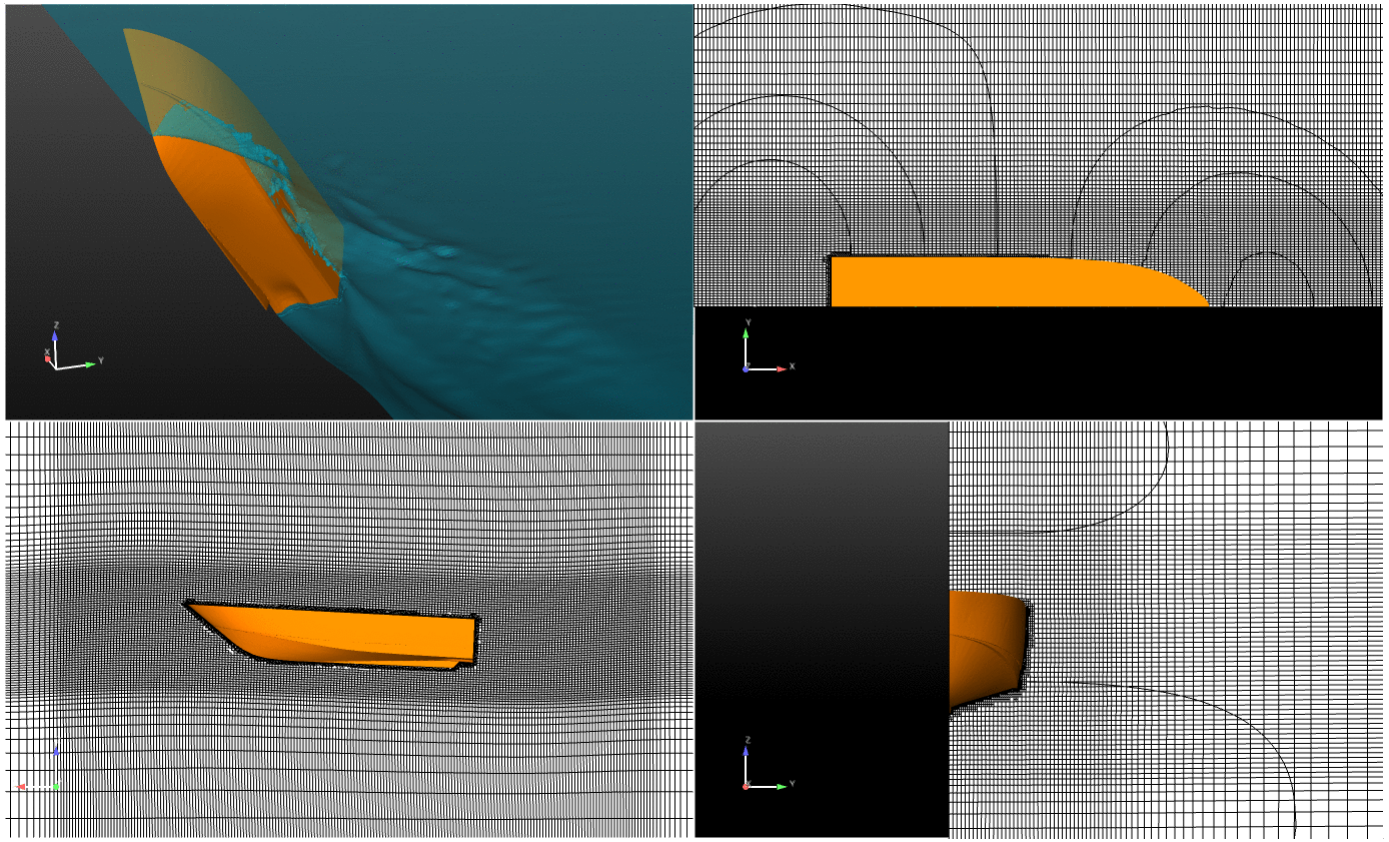


http://www.wolfdynamics.com/training/dynamicMeshes/relativeMotion.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Mesh morphing and rigid body motion – Sea keeping (VOF))

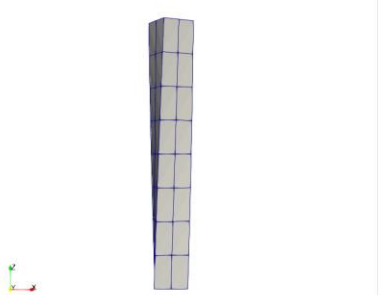


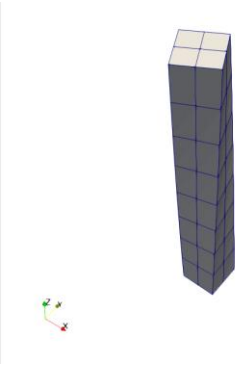
http://www.wolfdynamics.com/training/dynamicMeshes/seakeeping.gif

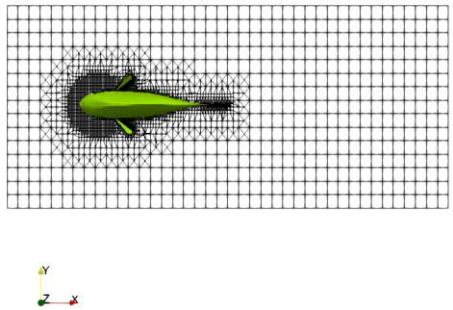
Introduction – What are dynamic meshes?

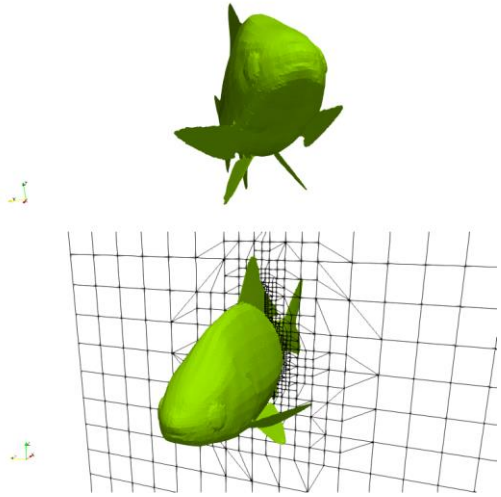
A few examples of dynamic meshes in OpenFOAM

Surface patches deformation with mesh smoothing









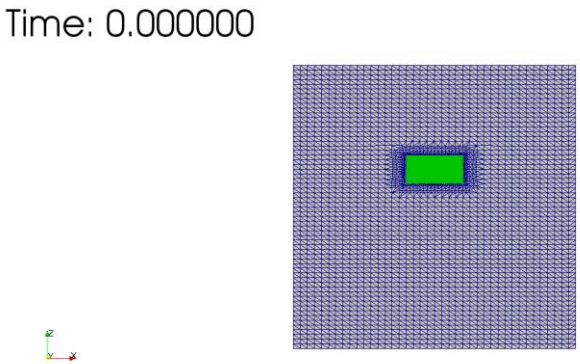
http://www.wolfdynamics.com/training/dynamicMeshes/twisting_column.gif

http://www.wolfdynamics.com/training/dynamicMeshes/fish_deforming.gif

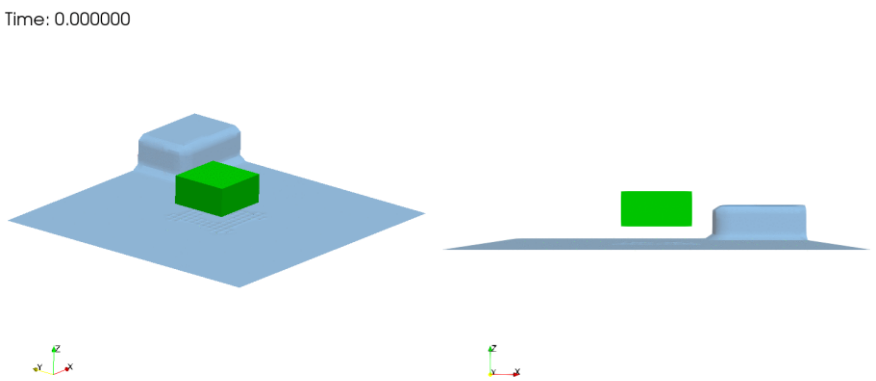
Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Mesh morphing together with mesh remeshing and rigid body motion.



http://www.wolfdynamics.com/training/dynamicMeshes/falling1.gif

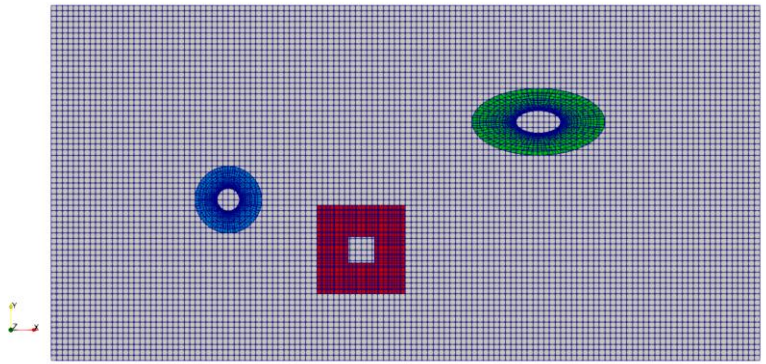


http://www.wolfdynamics.com/training/dynamicMeshes/falling2.gif

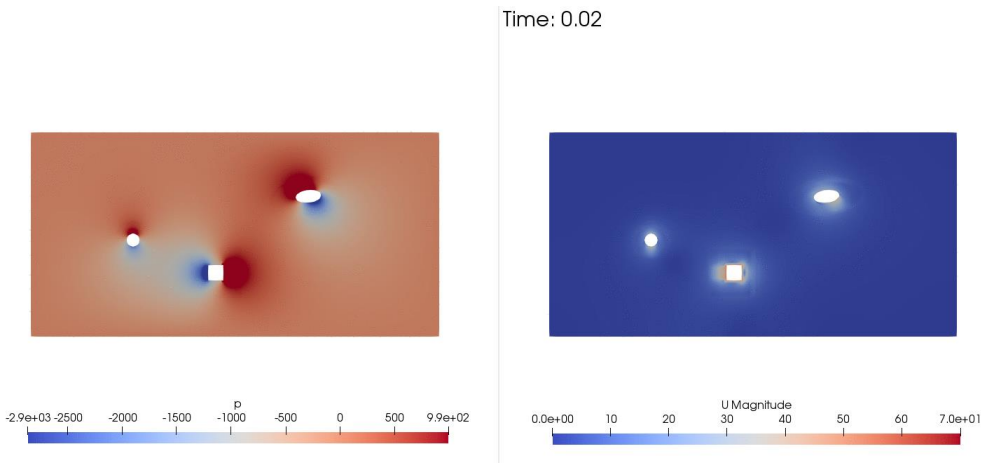
Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Overset meshes – Multiple bodies



http://www.wolfdynamics.com/training/dynamicMeshes/overset1.gif

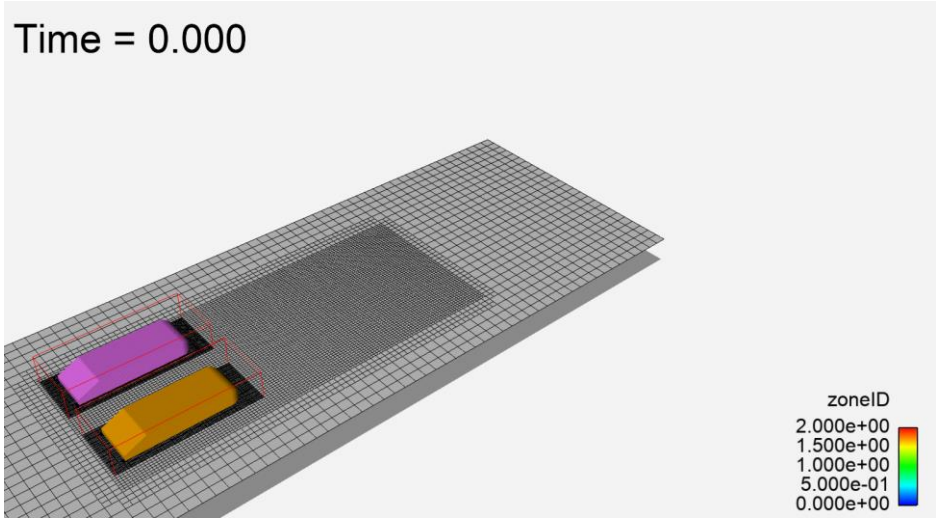


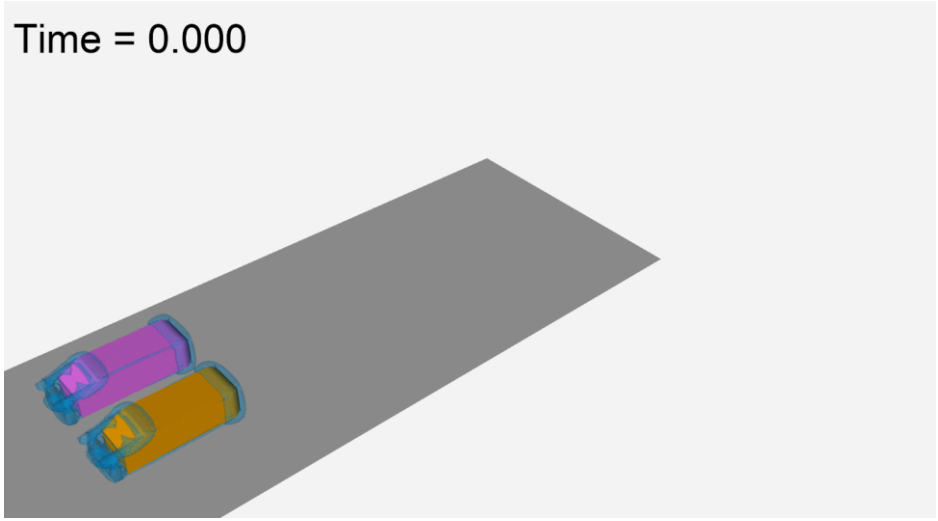
http://www.wolfdynamics.com/training/dynamicMeshes/overset2.gif

Introduction – What are dynamic meshes?

A few examples of dynamic meshes in OpenFOAM

Overset meshes – Overtaking car





http://www.wolfdynamics.com/training/dynamicMeshes/overtake1.gif

http://www.wolfdynamics.com/training/dynamicMeshes/overtake2.gif

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- In OpenFOAM, all dynamic meshes capabilities are controlled by a single dictionary, the *`dynamicMeshDict`* dictionary, which is located in the directory **`constant`**.
- Several types of dynamic meshes can be simulated in OpenFOAM, we will address the following cases:
    - Prescribed motion.
    - Rigid body motion.
    - Sliding meshes.
    - Adaptive mesh refinement (AMR).
- Setting moving bodies simulations is not so different from setting cases with fixed meshes.
- The main difference is that if the body is moving, we must assign a motion type to a surface patch, a cell region, or the whole domain.
- If we are using AMR, we must choose the refinement criterion based in a scalar field.

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- The dynamic meshes capabilities are selected in the dictionary *`constant/dynamicMeshDict`*.
- In OpenFOAM, there are several mesh motion solvers implemented, they deal with mesh motion displacement or mesh motion velocity.
- To name a few mesh motion solvers: **displacementLaplacian**, **displacementComponentLaplacian**, **displacementSBRStress**, **velocityLaplacian**.
- In the case of prescribed motion of a boundary patch, the motion is assigned in the dictionary *`0/pointDisplacement`* (if you use a mesh motion solver based on the mesh displacement), or in the dictionary *`0/pointMotionU`* (if you use a mesh motion solver based on the mesh velocity).
- In the case of rigid body motion, the motion is assigned in the dictionary *`0/pointDisplacement`*. The boundary condition of the moving patch is of the type **calculated**.
- Also, the boundary type of the moving walls must be **movingWallVelocity**, this is set in the dictionary *`0/U`*.
- Remember, you will need to adjust the numerics according to your physics.
- To use dynamic meshes capabilities, you will need to use solvers able to deal with dynamic meshes.

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- To find which solvers work with dynamic meshes, go to the solvers directory by typing `sol` in the command line interface. Then type in the terminal:
  - `$> grep -r dynamicFvMesh.H`

  The solvers that include the header file *`dynamicFvMesh.H`* support dynamic meshes.

- In OpenFOAM 8 (www.openfoam.org), the following solvers support dynamic meshes:
  - `pimpleFoam`, `interPhaseChangeDyMFoam`, `potentialFreeSurfaceDyMFoam`, `interMixingFoam`, `interFoam`, `multiphaseInterFoam`, `cavitatingDyMFoam`, `compressibleInterDyMFoam`, `PDRFoam`, `icoUncoupledKinematicParcelDyMFoam`, `sprayDyMFoam`, `uncoupledKinematicParcelDyMFoam`, `DPMDyMFoam`, `rhoCentralDyMFoam`, `rhoPimpleFoam`, `sonicDyMFoam`.

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- To find which solvers work with dynamic meshes, go to the solvers directory by typing `sol` in the command line interface. Then type in the terminal:
  - `$> grep -r dynamicFvMesh.H`

  The solvers that include the header file *`dynamicFvMesh.H`* support dynamic meshes.

- In OpenFOAM 8 (www.openfoam.org), the following solvers support dynamic meshes:
  - `PDRFoam`, `reactingFoam`, `rhoCentralFoam`, `rhoPimpleFoam`, `buoyantPimpleFoam`, `pimpleFoam`, `DPMFoam`, `particleFoam`, `rhoParticleFoam`, `sprayFoam`, `cavitatingFoam`, `compressibleInterFoam`, `interFoam`, `interMixingFoam`, `interPhaseChangeFoam`, `multiphaseInterFoam`, `potentialFreeSurfaceFoam`.

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- To find which solvers work with dynamic meshes, go to the solvers directory by typing `sol` in the command line interface. Then type in the terminal:
  - `$> grep -r dynamicFvMesh.H`

  The solvers that include the header file *`dynamicFvMesh.H`* support dynamic meshes.

- In OpenFOAM 2012 (www.openfoam.com), the following solvers support dynamic meshes:
  - `overLaplacianDyMFoam`, `overPotentialFoam`, `PDRFoam`, `XiDyMFoam`, `rhoCentralDyMFoam`, `overRhoPimpleDyMFoam`, `rhoPimpleFoam`, `overRhoSimpleFoam`, `sonicDyMFoam`, `overBuoyantPimpleDyMFoam`, `solidFoam`, `overPimpleDyMFoam`, `pimpleFoam`, `overSimpleFoam`, `DPMDyMFoam`, `icoUncoupledKinematicParcelDyMFoam`, `reactingParcelFoam`, `sprayDyMFoam`, `uncoupledKinematicParcelDyMFoam`, `cavitatingDyMFoam`, `compressibleInterDyMFoam`, `compressibleInterIsoFoam`, `interCondensatingEvaporatingFoam`, `interFoam`, `interMixingFoam`, `overInterDyMFoam`, `interIsoFoam`, `interPhaseChangeDyMFoam`, `multiphaseInterFoam`, `potentialFreeSurfaceDyMFoam`.

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- You will find the source code of all the mesh motion libraries in the directories:
  - **`OpenFOAM-8/src/dynamicFvMesh`**
  - **`OpenFOAM-8/src/dynamicMesh`**
  - **`OpenFOAM-8/src/fvMotionSolver`**
  - **`OpenFOAM-8/src/rigidBodyDynamics`**
  - **`OpenFOAM-8/src/rigidBodyMeshMotion`**
  - **`OpenFOAM-8/src/rigidBodyState`**
  - **`OpenFOAM-8/src/sixDoFRigidBodyMotion`**
  - **`OpenFOAM-8/src/sixDoFRigidBodyState`**
  - **`OpenFOAM-8/src/topoChangerFvMesh`**

- You will find the source code of the prescribed patch motions in the directory:
  - **`OpenFOAM-8/src/fvMotionSolver/pointPatchFields/derived`**

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- You will find the source code of the restraints/constraints of the rigid body motion solvers (6DoF and rigid body dynamics approaches) in the directory:
  - **`OpenFOAM-8/src/sixDoFRigidBodyMotion/sixDoFRigidBodyMotion`**
  - **`OpenFOAM-8/src/rigidBodyDynamics`**

- You will find the source code of the mesh diffusivity models in the directory:
  - **`OpenFOAM-8/src/fvMotionSolver/motionDiffusivity`**

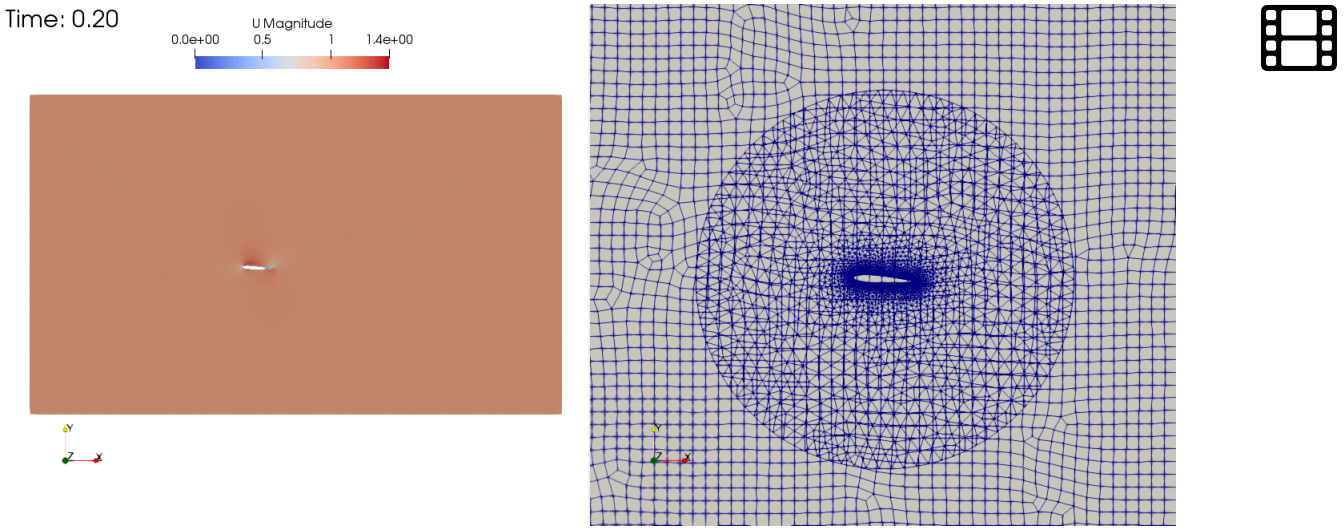
- You will find the source code of the mesh motion solvers in the directory
  - **`OpenFOAM-8/src/fvMotionSolver/fvMotionSolvers`**

- If you are using version 2012, you will find the libraries in the same locations, but instead of the directory **`OpenFOAM-8`**, they will be in the directory **`OpenFOAM-v2012,`** that is:
  - **`OpenFOAM-v2012/src/...`**

Introduction – What are dynamic meshes?

Dynamic meshes in OpenFOAM

- Additionally, OpenFOAM gives you overset meshes capabilities.
- However, these capabilities are only available in the version developed by ESI-OpenCFD (www.openfoam.com).
- The latest version is v2012. You can find the releases notes in the following link: https://www.openfoam.com/releases/openfoam-v2012/
- You will find the source code of the overset library in the directory:
    - **`OpenFOAM-v2012/src/overset`**



http://www.wolfdynamics.com/training/dynamicMeshes/overset3.gif

Introduction – What are dynamic meshes?

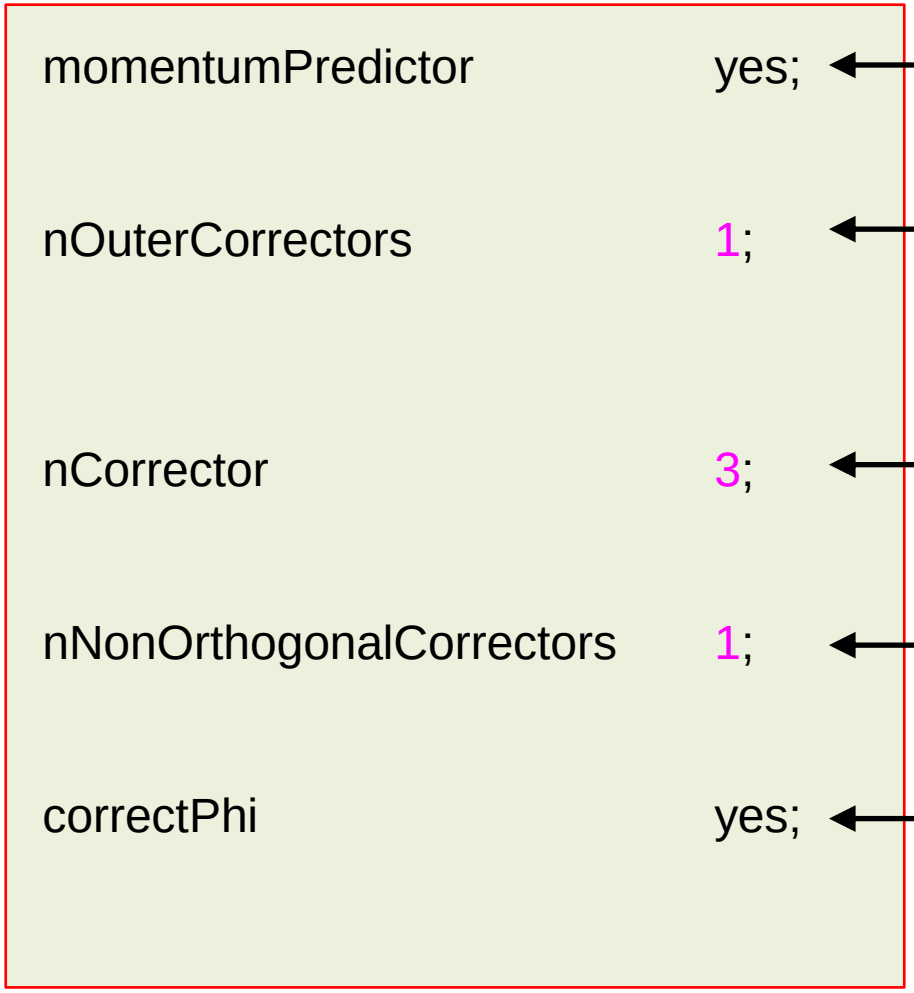
A short note on the numerics required for dynamic meshes in OpenFOAM

- Dynamic meshes have tighter stability and accuracy requirements than static meshes.
- During simulations with moving bodies, the boundaries/mesh/domain will experience strong instantaneous accelerations, large displacements/deformations, and fluctuations in linear/angular velocities.
    - This requires the use of a robust and accurate numerics.
- During AMR simulations, the meshes will change, cells will become smaller and smaller (hence cell's volume), therefore, the CFL number will change from refinement level to refinement level.
    - Again, this requires the use of a robust and accurate numerics.
- It is extremely important to control the CFL number and be sure that it remains constant within the limits of stability.
- It is also recommended to always monitor the mesh quality when dealing with morphing meshes and adjust the numerical method accordingly.
- No need to say that dynamic meshes are intrinsically unsteady.
- And that they are computationally expensive and very time consuming.

Introduction – What are dynamic meshes?

A short note on the numerics required for dynamic meshes in OpenFOAM

- For the *fvSolution* dictionary:



Set to yes for high Reynolds flows, where convection dominates (default value is yes)

You should do at least 1 corrector step (equivalent to PISO). If you are dealing with moving bodies, LES simulations, or if the CFL number is higher than 1, do a minimum of 2 correctors. For best results (specially with moving bodies), do at least 5 correctors.

Recommended to use at least 3 correctors. It improves accuracy and stability. Use 4 or more for highly transient flows or strongly coupled problems.

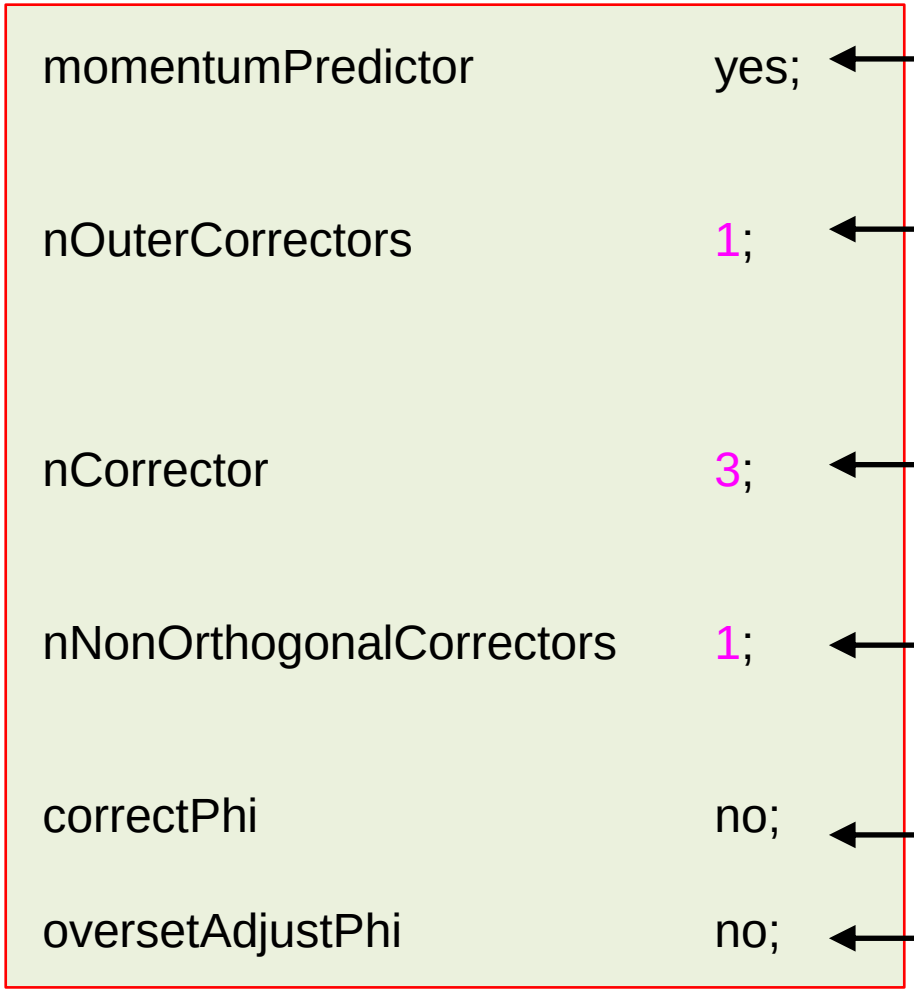
Recommended to use at least 1 corrector. Increase the value for bad quality meshes or if you expect large mesh deformations.

Flux corrections to ensure continuity. Default value is yes. Required during start-up, restart, mesh-motion, etc., when non-conservative fluxes may adversely affect the solution. This is particularly important for VoF and other multi-phase solver in which non-conservative fluxes cause unboundedness of the phase-fraction.

Introduction – What are dynamic meshes?

A short note on the numerics required for dynamic meshes in OpenFOAM

- For the *fvSolution* dictionary and if you are dealing with overset meshes:



Set to yes for high Reynolds flows, where convection dominates (default value is yes)

You should do at least 1 corrector step (equivalent to PISO). If you are dealing with moving bodies, LES simulations, or if the CFL number is higher than 1, do a minimum of 2 correctors. For best results (specially with moving bodies), do at least 5 correctors.

Recommended to use at least 3 correctors. It improves accuracy and stability. Use 4 or more for highly transient flows or strongly coupled problems..

Recommended to use at least 1 corrector. Increase the value for bad quality meshes or if you expect large mesh deformations.

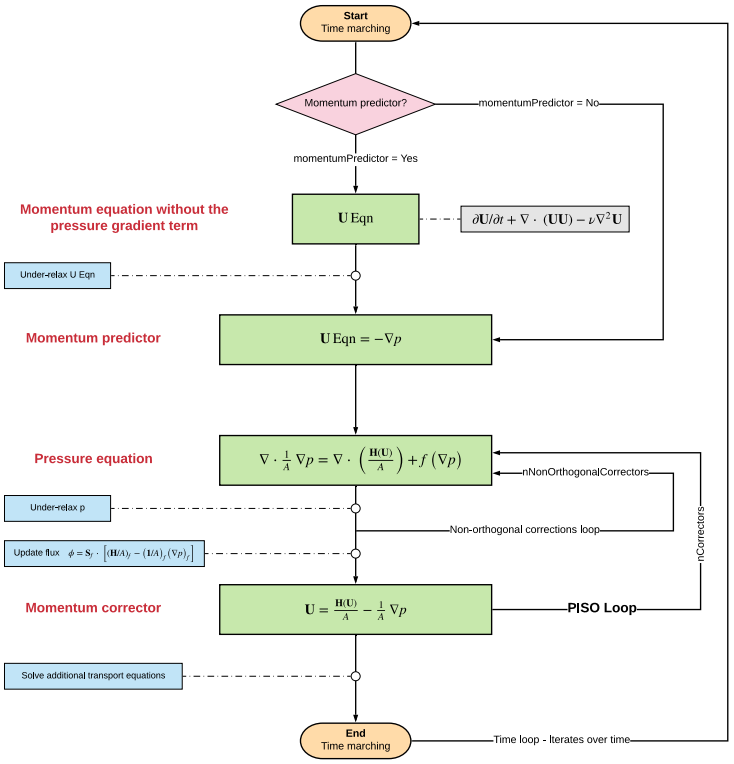
Flux corrections. In overset meshes the default value is no. It is recommended to leave it always off (no) in overset meshes.

Flux corrections in overset meshes. Use with incompressible flows in closed domains and if you experience pressure fluctuations. Default value is no.

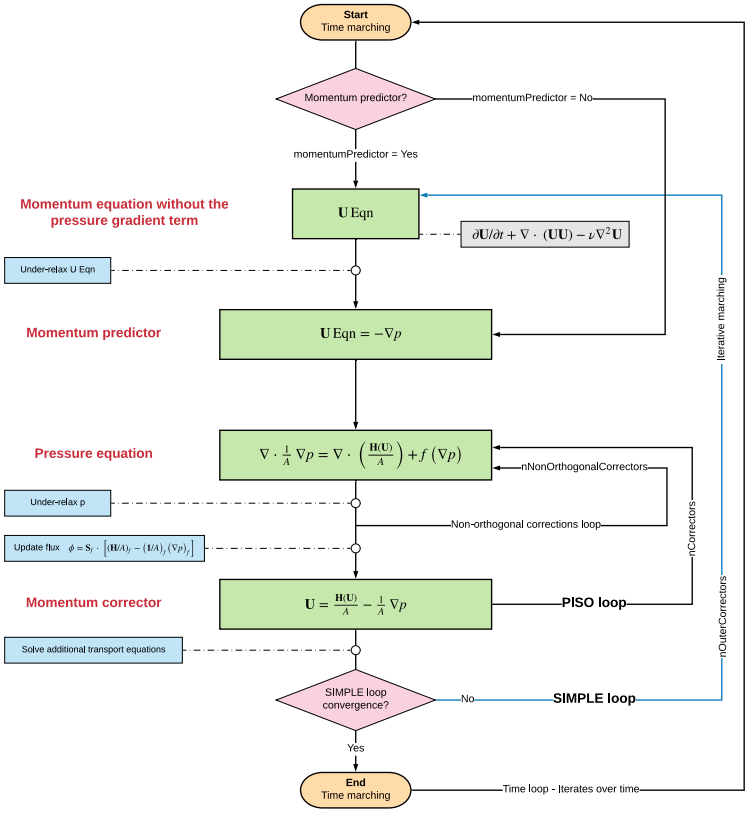
Introduction – What are dynamic meshes?

A short note on the numerics required for dynamic meshes in OpenFOAM

- In OpenFOAM, there are two types of unsteady loops, standard **PISO** (non-iterative marching or **NITA**), and **PIMPLE** (**PISO** with iterative marching or **ITA**).
- When dealing with moving bodies or overset meshes, it is extremely recommended to use the **PISO-ITA** method (or **PIMPLE**) with at least two outer correctors.



PISO-NITA (non-iterative marching)



PISO-ITA (iterative marching) - PIMPLE

Introduction – What are dynamic meshes?

A short note on the numerics required for dynamic meshes in OpenFOAM

- For the *fvSchemes* dictionary (overset and body fitted meshes)

```
ddtSchemes
{
    default         Euler;
}

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

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwindV grad(U);
}

laplacianSchemes
{
    default         Gauss linear limited 0.5;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         limited 0.5;
}
```

For good accuracy and stability use the Euler method with a CFL of 0.9 or less. You can use backward or CrankNicolson but they are unbounded and might give stability problems (to stabilize the solution you will need to use a CFL of 0.9 or less).

For stability, use slope limiters for gradients. The leastSquares method is more accurate but a little bit oscillatory for bad quality meshes.

For good accuracy use at least second order accurate methods. The second order upwind method is a good choice for the momentum term (stable and accurate).

For good quality meshes (non orthogonality less than 70), use limited 1. For industrial meshes with large orthogonality (more than 70), it is recommended to use limited 0.5.

This entry refers to the method used to interpolate values from cell centers to face centers. It is unlikely that you will need to use something different from linear.

Use same option as for laplacianSchemes

Introduction – What are dynamic meshes?

A short note on the numerics required for dynamic meshes in OpenFOAM

- As the mesh is changing in dynamic meshes, it is advisable to fix the CFL number.
- To ensure that we have a constant CFL number or we do not exceed a given CFL value as the mesh changes, we can use adaptive time stepping (**adjustableTimeStep**).
- This option is set in the dictionary *controlDict*, and it is supported by all solvers with dynamic meshes capabilities.
- However, it is better to fix the time-step (to avoid oscillations introduced due to the adaptive time-stepping), and keep the CFL number below one, but this may be difficult to achieve.

```
...
...
...

    adjustTimeStep  yes;

    maxCo           1;

    maxAlphaCo      0.5;

    maxDeltaT       0.1;

...
...
...
```

Enable/disable adjustable time step

Maximum allowable CFL number.
The solver will automatically adjust the time-step so it does not exceed this limit

Maximum allowable CFL number for the volume fraction alpha. Only available when using multiphase solvers.

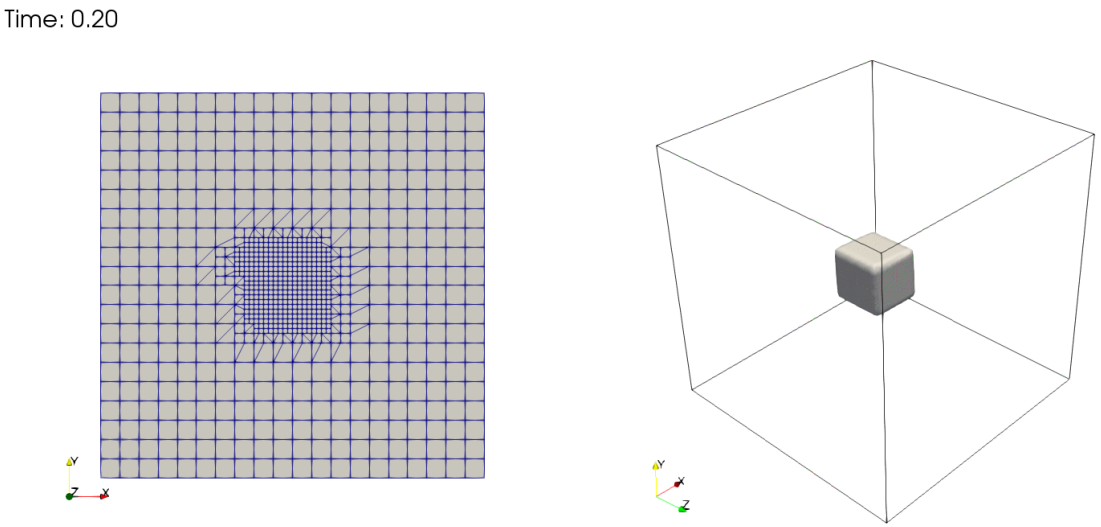
Maximum time-step

Roadmap

1. ~~Introduction – What are dynamic meshes?~~
2. **Adaptive mesh refinement in OpenFOAM**
3. Sliding meshes in OpenFOAM
4. Morphing meshes in OpenFOAM
5. Moving meshes in OpenFOAM
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

Adaptive mesh refinement in OpenFOAM

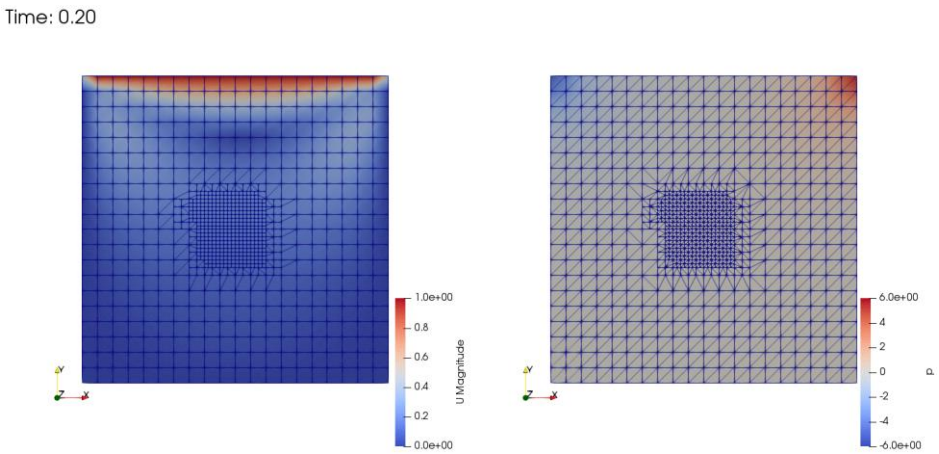
- Adaptive mesh refinement or AMR, consist in automatically refining the mesh according to a predefined criterion and given control parameters (refinement levels, unrefinement levels, expansion ratio and so on).
- AMR in OpenFOAM is compatible with all solvers supporting dynamic meshes capabilities and it is only supported for hexahedral meshes.
- If you need 2D AMR capabilities and/or more advanced AMR capabilities (such as dynamic load balancing or multiple refinement criteria), you can install the library **dynamicloadbalancing** developed by the Technical University Darmstadt (https://bitbucket.org/dynamicloadbalancing/dynamicloadbalancing).



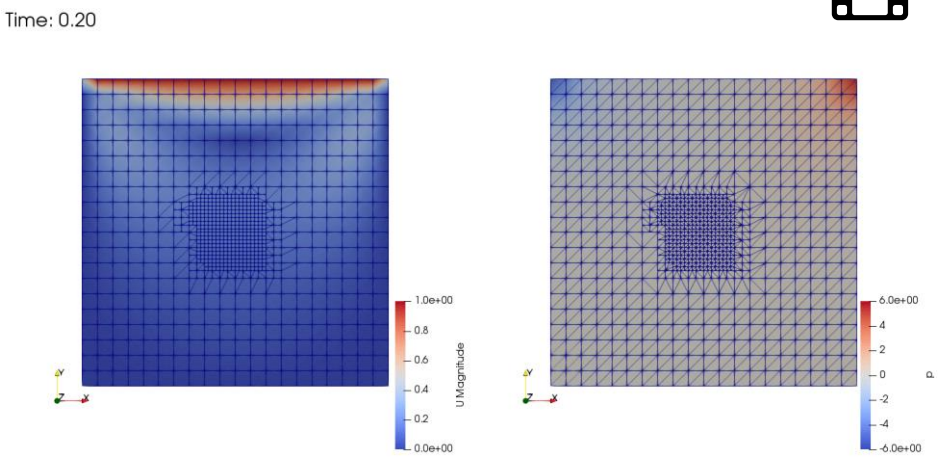
Passive scalar tracking using AMR
http://www.wolfdynamics.com/training/dynamicMeshes/amr3.gif

Adaptive mesh refinement in OpenFOAM

- When using AMR, it is particularly important to use an accurate and stable numerical method.
- If you use a method that it is unbounded and/or too diffusive (in space and time), the AMR approach will not be able to fully track the quantity of interest, and you may have overshoots/undershoots in your solution.
- The easiest way too control the time step in OpenFOAM as the mesh is refined, is by using the **adjustableTimeStep** option (available with the pimple family solvers) and a CFL lower than 0.8 (we recommend a value of 0.5).
- AMR adds considerable overhead to the computations.



Maximum CFL = 0.5
http://www.wolfdynamics.com/training/dynamicMeshes/amr4.gif



Maximum CFL = 1.0
http://www.wolfdynamics.com/training/dynamicMeshes/amr5.gif

Adaptive mesh refinement in OpenFOAM

- Let us run the 3D driven cavity with AMR case.
- You will find this case in the directory:

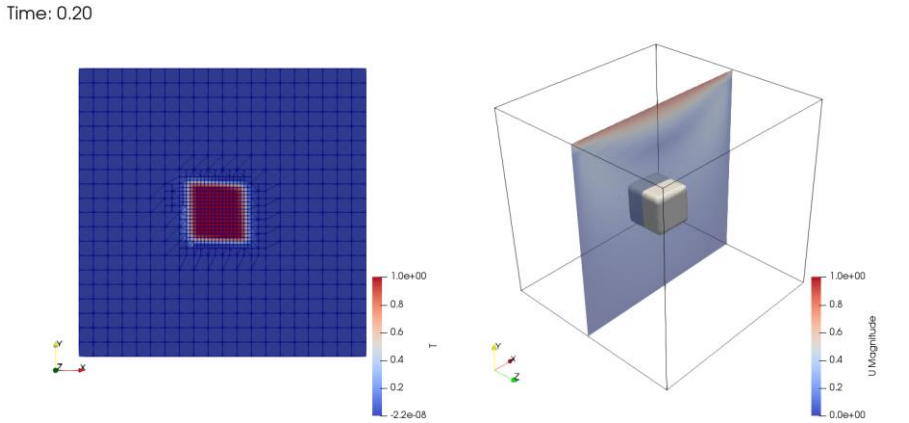
```
$TM/AMR_mesh/cavity3d
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

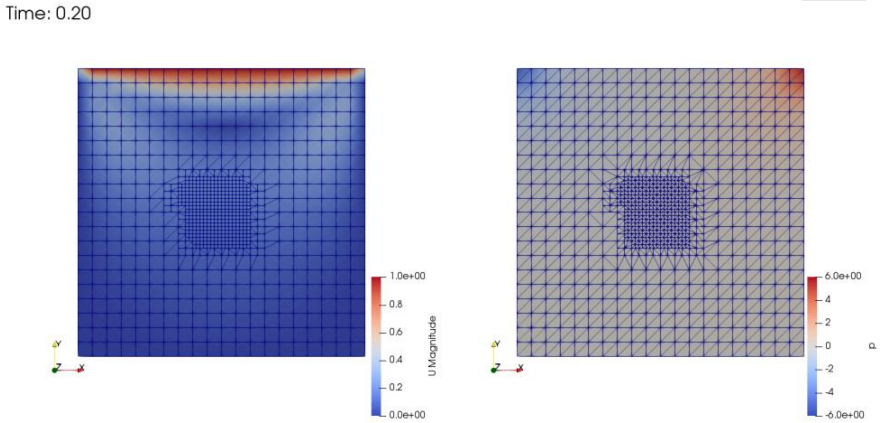
Adaptive mesh refinement in OpenFOAM

3D driven cavity with AMR

- Hereafter we will use the driven cavity case to introduce the AMR capabilities available in OpenFOAM.
- Remember, all dynamic mesh capabilities are controlled in the dictionary `dynamicMeshDict`, this dictionary is located in the directory **`constant`**.
- In this case, we will initialize a passive scalar which we will use as the base field for the AMR.
- It is important to mention that AMR works only with scalar fields.



http://www.wolfdynamics.com/training/dynamicMeshes/amr1.gif



http://www.wolfdynamics.com/training/dynamicMeshes/amr2.gif

Adaptive mesh refinement in OpenFOAM

3D driven cavity with AMR

- Let us open the *dynamicMeshDict* dictionary.

```
...
...
...

dynamicFvMesh   dynamicRefineFvMesh;


dynamicRefineFvMeshCoeffs
{
        refineInterval  1;

        field           T;

        lowerRefineLevel 0.4;
        upperRefineLevel 1;

        unrefineLevel   10;

        nBufferLayers   1;

        maxRefinement   2;

        maxCells        100000;

...
...
...
```

Library with AMR capabilities

How often refine/unrefine (time-step or iteration)

Quantity of interest

Refine the field between these two levels

If field value < unrefineLevel unrefine

Number of layer between refinement levels

Maximum refinement level (1 means no refinement)

Stop refinement if the maximum number of cells is reached

Adaptive mesh refinement in OpenFOAM

3D driven cavity with AMR

- Let us open the *dynamicMeshDict* dictionary.

```
...
...
...

        correctFluxes
        (
                ( phi U )
                //( phi none )
                //( alphaPhi none )
         );


        dumpLevel      true;

}
```

Flux field and corresponding velocity field. Fluxes on changed faces get recalculated by interpolating the velocity.
Use 'none' on surfaceScalarFields that do not need to be reinterpolated.

Write the refinement level as a scalar field

Adaptive mesh refinement in OpenFOAM

3D driven cavity with AMR

- This case is ready to run, we are going to work in parallel.
- To run the case, in the terminal window type:

1. `$> foamCleanTutorials`
2. `$> blockMesh`
3. `$> cp 0/T.org 0/T`
4. `$> setFields`
5. `$> decomposePar`
6. `$> mpirun -np 4 renumberMesh -parallel -overwrite | tee log.renumberMesh`
7. `$> mpirun -np 4 pimpleFoam -parallel | tee log.solver`
8. `$> paraFoam -builtin`

3D driven cavity with AMR

- To reconstruct the parallel case, we proceed as follows:

1. `$> reconstrucParMesh`
2. `$> reconstructPar`
3. `$> paraFoam`

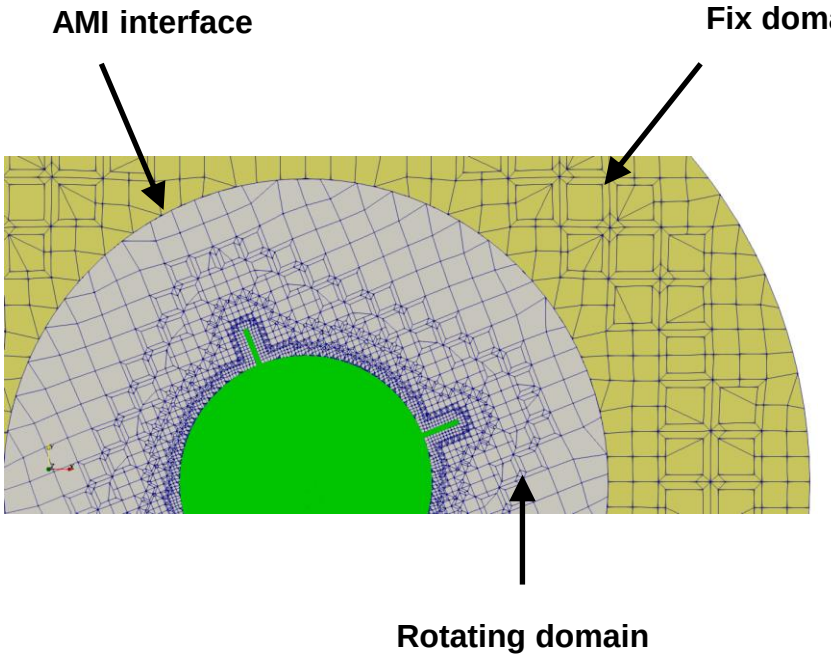
- In step 1, we reconstruct the refined mesh. This step is compulsory in AMR as the number of cells and connectivity of the mesh have been changed.
- In step 2, we reconstruct the solution.
- In step 3, we can visualize the reconstructed solution (single processor).

Roadmap

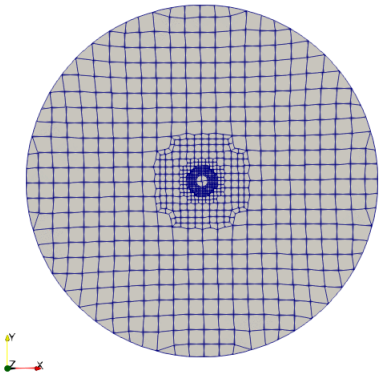
1. ~~Introduction – What are dynamic meshes?~~
2. ~~Adaptive mesh refinement in OpenFOAM~~
3. **Sliding meshes in OpenFOAM**
4. Morphing meshes in OpenFOAM
5. Moving meshes in OpenFOAM
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

Sliding meshes in OpenFOAM

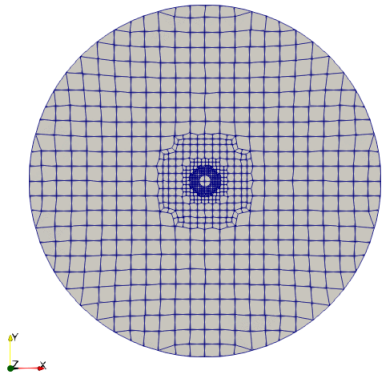
- In sliding meshes simulations, the solution is interpolated back-and-forth between topologically separated regions.
- The interpolation is done at the mesh interface. In OpenFOAM, this patch type is called arbitrary mesh interface or AMI.
- To reduce interpolation errors at the AMI patches, the meshes should be similar in the master and slave patches.



http://www.wolfdynamics.com/training/movingbodies/image8.gif



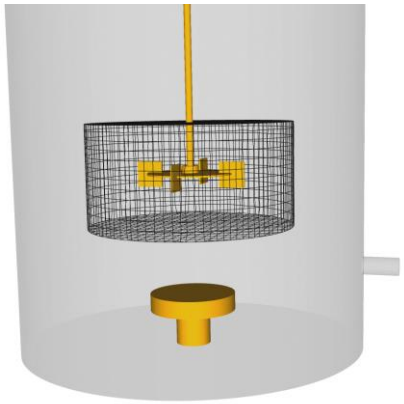
Rotating patch
Master patch

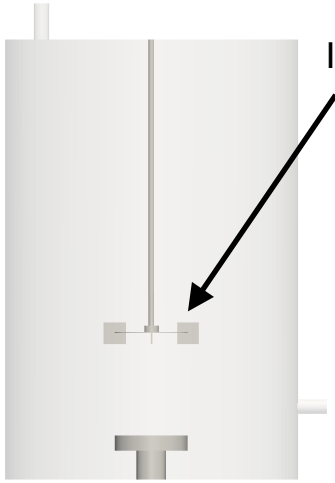


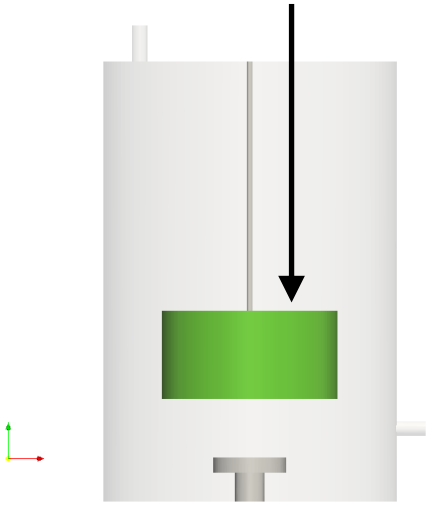
Fix patch
Slave patch

Sliding meshes in OpenFOAM

- The different regions (*i.e.*, fix and rotating), must be created at meshing time.
- The mesh can be generated using OpenFOAM meshing capabilities or any external mesher.
- Single and multiple rotating bodies are supported.
- Sliding meshes are compatible with all physical modeling capabilities implemented in OpenFOAM.



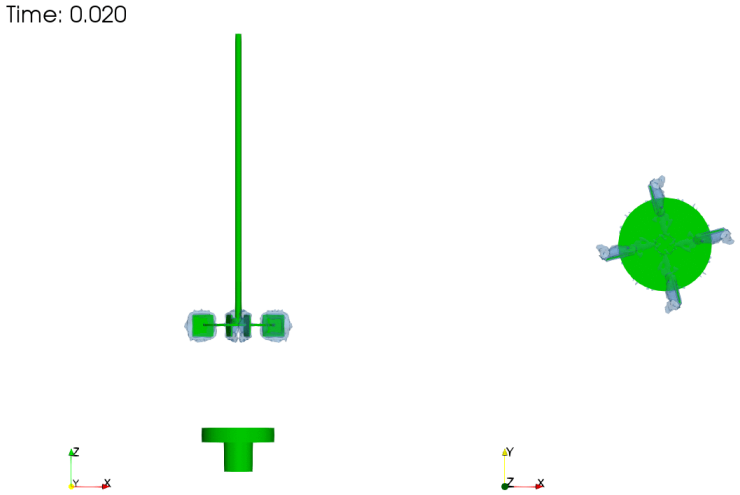




http://www.wolfdynamics.com/training/meshing/image5.gif

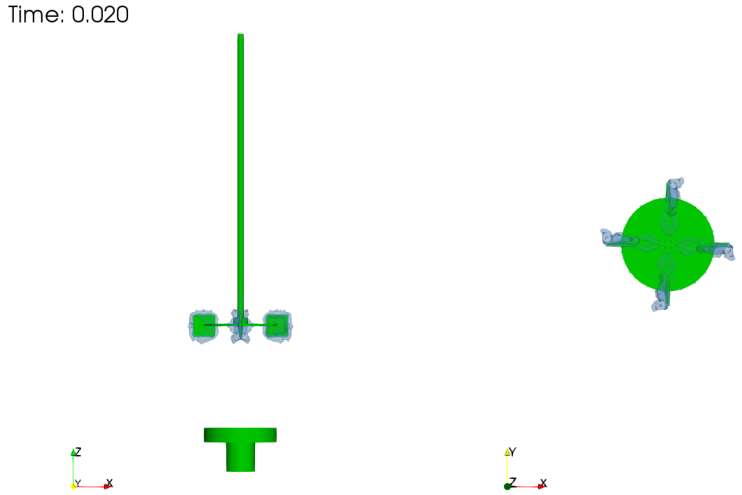
Sliding meshes in OpenFOAM

- Sliding meshes are intrinsically unsteady.
- No need to say that in order to use sliding meshes you need axial symmetry.
- An alternative to sliding meshes is MRF (steady and unsteady).
- In the MRF approach, the mesh is fixed. The rotation is accounted by adding source terms to the region of interest.
- MRF solvers are mush faster than sliding meshes solvers.



Sliding grids – Unsteady solver

http://www.wolfdynamics.com/training/movingbodies/image13.gif



MRF – Steady solver

http://www.wolfdynamics.com/training/movingbodies/image14.gif

Sliding meshes in OpenFOAM

- Let us run the continuous stirring tank reactor case.
- You will find this case in the directory:

```
$TM/sliding_MRF_meshes/CSTR/sliding_piso/
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

CSTR – Continuous stirring tank reactor mesh

- Let us open the *snappyHexMesh* dictionary.
- Explaining the whole dictionary is outside of the scope of this training, we will focus our attention in the section where we create the different zones (a cell zone and a face zone).
- Let us take a look at the **castellatedMeshControls** section of the dictionary *snappyHexMeshDict*. In this block we define the cellZone and faceZone, as follows,

```
castellatedMeshControls
{
        ...
        ...
        ...

        //Surface based refinement
        inner_volume
        (
                level (1 1);

                cellZone cell_inner_volume;

                faceZone face_inner_volume;

                cellZoneInside insidePoint;

                insidePoint (50 0 100);
        );
        ...
        ...
        ...
}
```

Using the surface inner_volume geometry (that was loaded in the geometry section of the snnapyHexMeshDict dictionary), we create a mesh zone that we will use at a later time to split the whole mesh in two regions.

Name of the cellZone.

Name of the faceZone.

Use an inner point to define location of the zone

Location of the insidePoint.
The point is located inside the surface inner_volume, therefore the new zone is created inside the surface selected.

CSTR – Continuous stirring tank reactor mesh

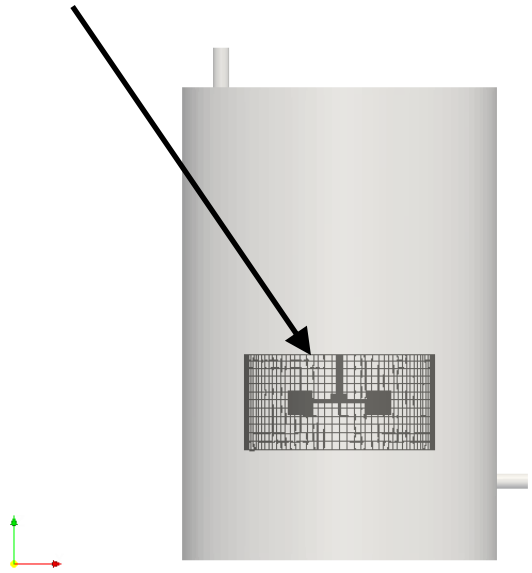
- At this point, we are going to work in parallel.
- To generate the mesh, in the terminal window type:

```
1. $> foamCleanTutorials
2. $> foamCleanPolymesh
3. $> surfaceFeatureExtract
4. $> blockMesh
5. $> decomposePar
6. $> mpirun -np 4 snappyHexMesh -parallel -overwrite
7. $> mpirun -np 4 checkMesh -parallel -latestTime
8. $> reconstructParMesh -constant
9. $> paraFoam
```

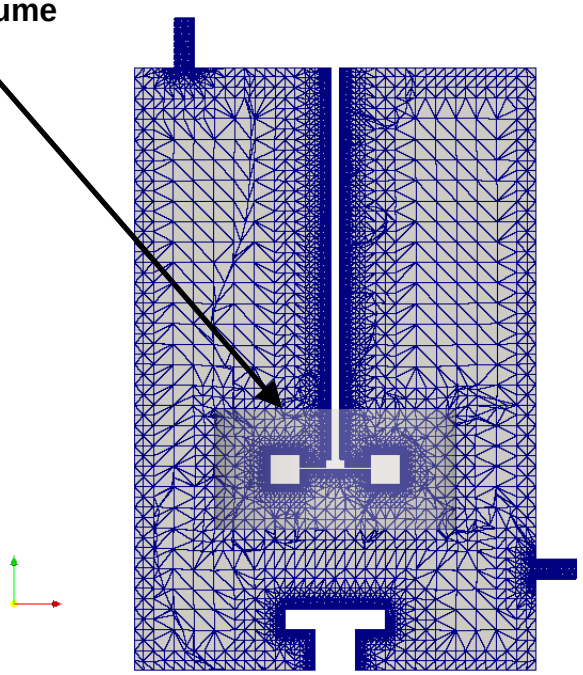
CSTR – Continuous stirring tank reactor mesh

- Using `paraFoam` let us take a look at the newly created mesh, including the zones (face and cells).

face_inner_volume



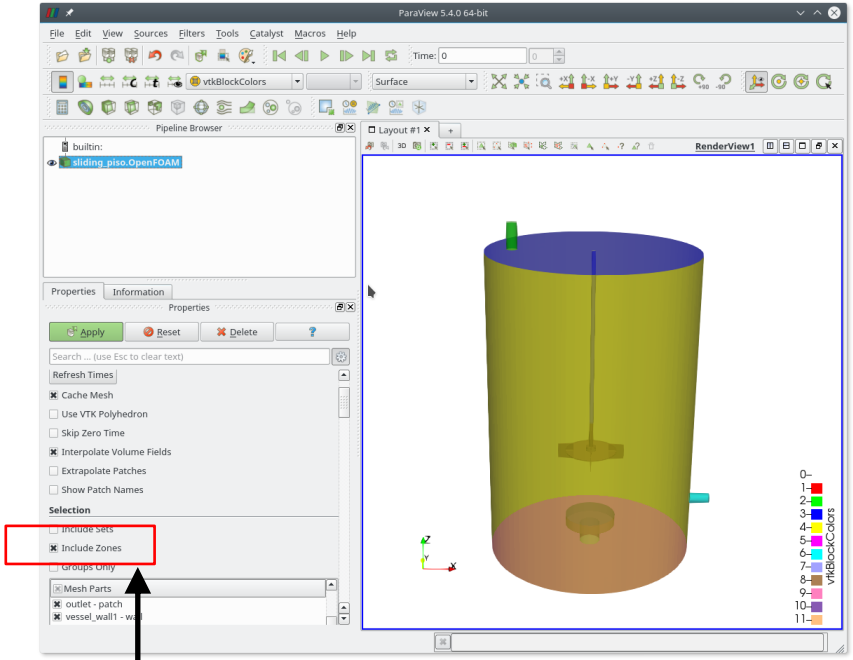
cell_inner_volume

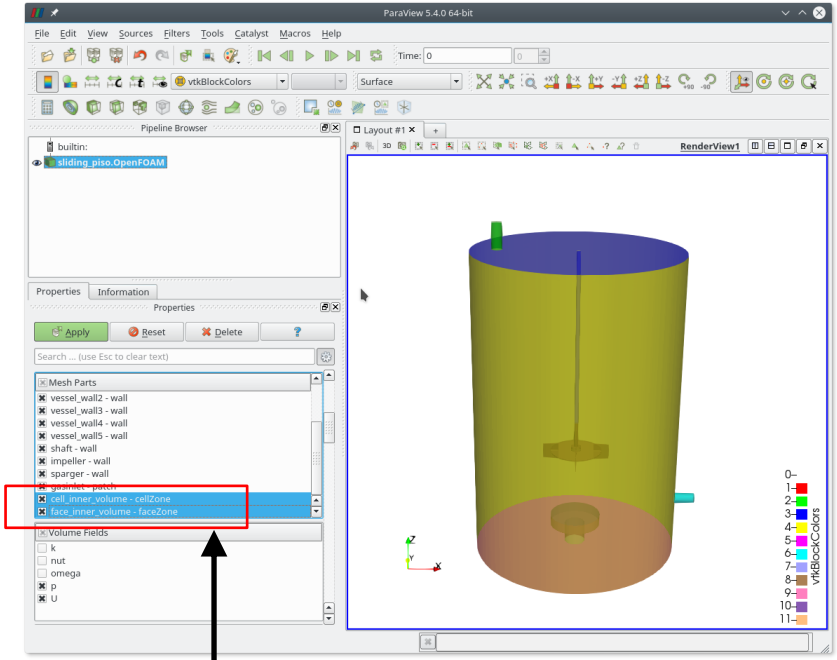


Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh

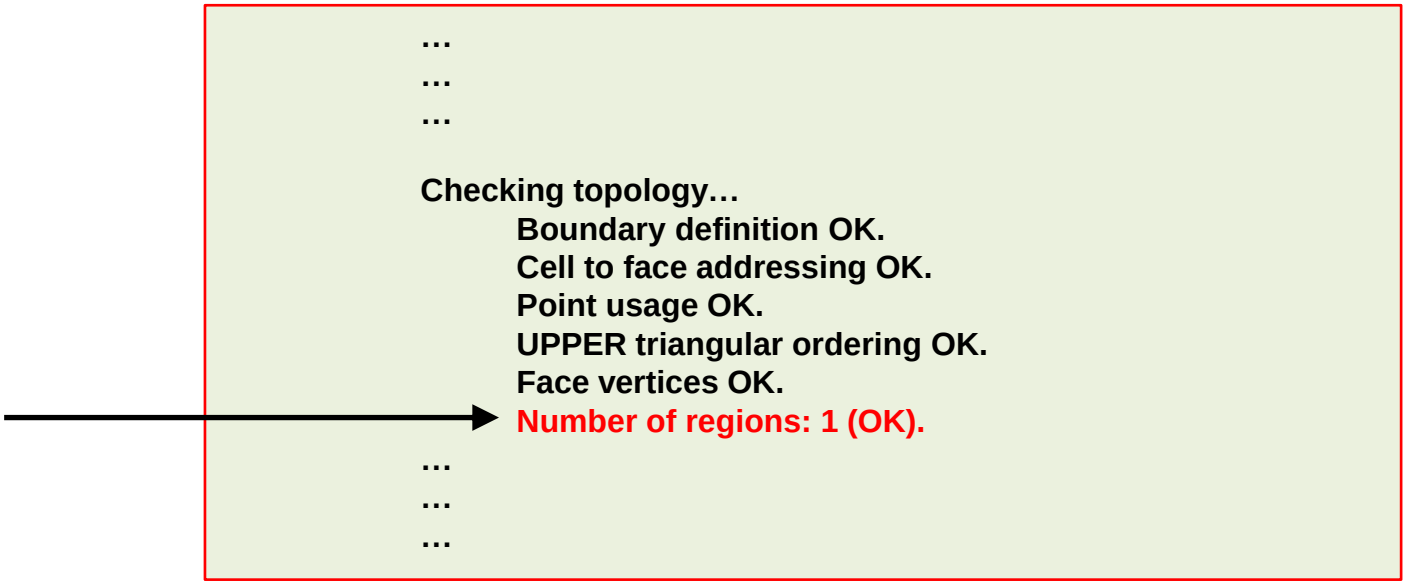
- To visualize the zones in `paraFoam` you will need to enable the option `Include Zones`
- Then select the mesh parts **cell_inner_volume** and **face_inner_volume**.





CSTR – Continuous stirring tank reactor mesh

- At this point and if you run `checkMesh`, you will get the following information:
  - `$> checkMesh`

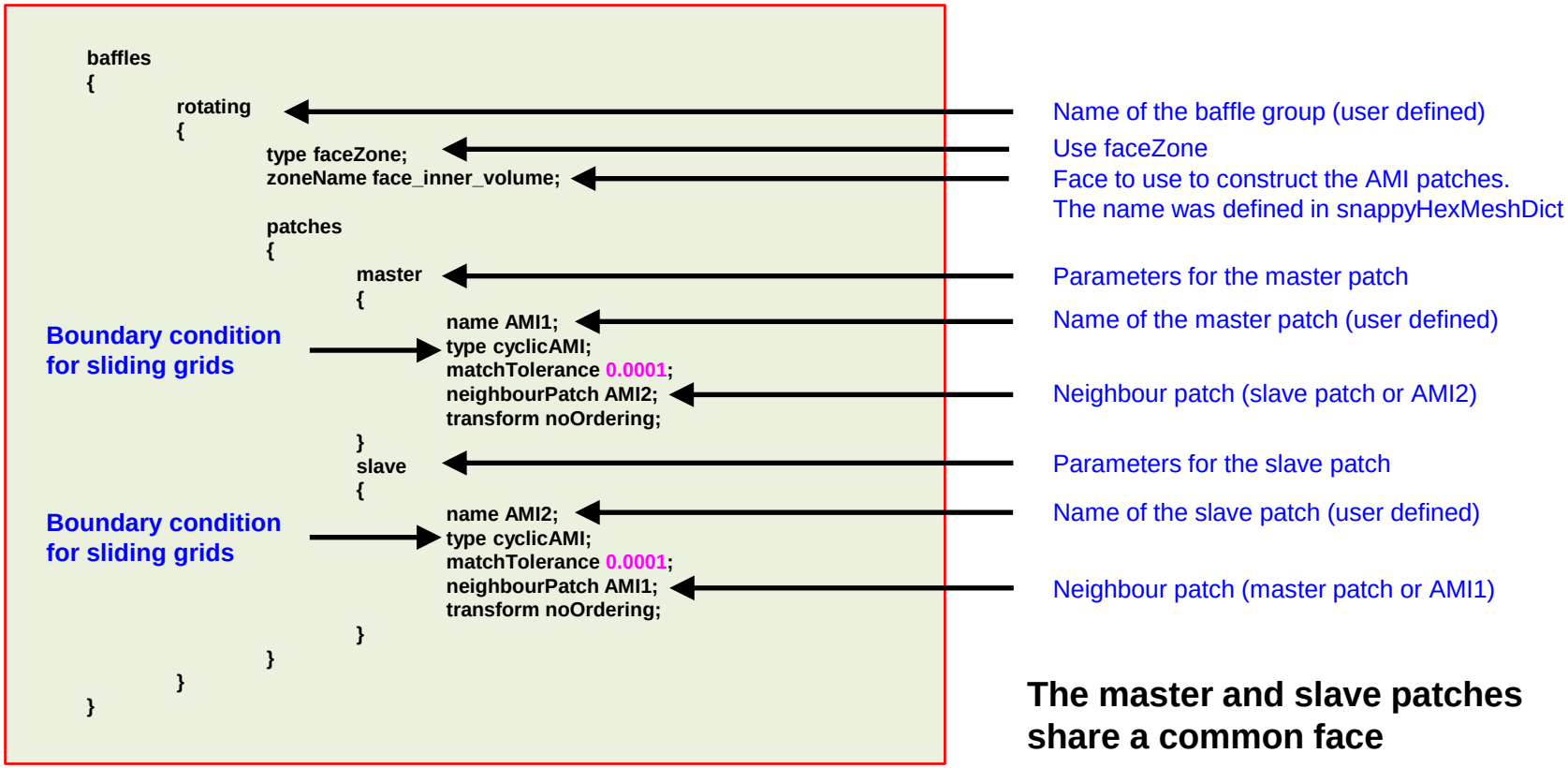


- As you can see we only have one region, but we are interested in having two regions.
- Up to this point the mesh is valid to be used with the MRF approach.
- For sliding meshes, we still need to split the mesh in two or more regions.
- We will use the following utilities:
  - `createBaffle`
  - `mergeOrSplitBaffles`

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh

- The utility `createBaffles`, reads the dictionary *`createBafflesDict`*.
- With this utility we create the interface patches between the fix zone and the rotating zone.



Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh

- To create the two regions, we proceed as follows (notice that we are going to work in serial from now on):

1. `$> createBaffles -overwrite`
2. `$> mergeOrSplitBaffles -split -overwrite`
3. `$> createPatch -overwrite`
4. `$> splitMeshRegions -makeCellZones -overwrite`
5. `$> splitMeshRegions -detectOnly`
6. `$> transformPoints -scale '(0.01 0.01 0.01)'`

- Steps 3-6 are optional.

CSTR – Continuous stirring tank reactor mesh

- So what did we do?
  - Step 1:
    - Splits the mesh in regions using the baffles (**faceZone**), created during the meshing stage.
    - We also create the **cyclicAMI** patches **AMI1** and **AMI2**.
    - At this point we have two regions and one zone. However, the two regions are stich together via the patches **AMI1** and **AMI2**.
  - Step 2: topologically split the patches **AMI1** and **AMI2**. As we removed the link between **AMI1** and **AMI2**, the regions are free to move.
  - Step 3 (optional): gets rid of zero faced patches if hey exist. These are the patches remaining from the base mesh, as they are empty, we do not need them.
  - Step 4 (optional):
    - Splits mesh into multiple zones. It will create automatically the sets and zones.
    - At this point we have two regions and two zones.
  - Step 5 (optional): just to show the regions and names.
  - Step 6 (optional): scales the mesh.

CSTR – Continuous stirring tank reactor mesh

- At this point and if you run `checkMesh`, you will get the following information:
  - `$> checkMesh`

```
…
…
…
Checking topology…
        Boundary definition OK.
        Cell to face addressing OK.
        Point usage OK.
        UPPER triangular ordering OK.
        Face vertices OK.
        *Number of regions: 2
        The mesh has multiple regions which are not connected by any face.
        <<Writing region information to ”0/cellToRegion”
        <<Writing region 0 with 136187 cells to cellSet region0
        <<Writing region 1 with 67682 cells to cellSet region1
…
…
…
```

Regions name

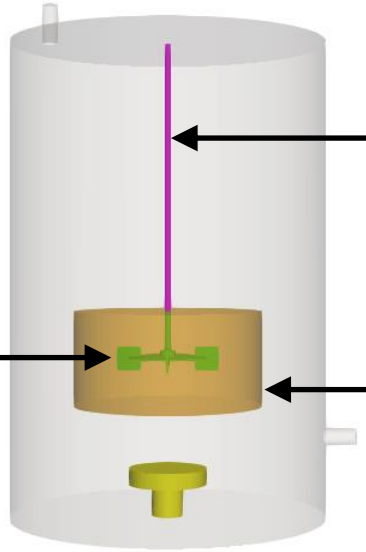
- As you can see, we now have two regions.
- At this point the mesh is ready to use.
- You can visualize the mesh (with all the sets and zones) using `paraFoam`.

CSTR – Continuous stirring tank reactor mesh

- At this point the mesh is ready to use using the sliding meshes approach. You can visualize the mesh using `paraFoam`.
- If you use `checkMesh`, it will report that there are two regions.
- In the dictionary *`constant/dynamicsMeshDict`* we set which region will move and the rotation parameters.
- To preview the region motion, in the terminal type:
    - `$> moveDynamicMesh`
- To preview the region motion and check the quality of the AMI interfaces, in the terminal type:
    - `$> moveDynamicMesh -checkAMI -noFunctionObjects`
- In our YouTube channel you can find an step-by-step video explaining this case.

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh



Shaft
type rotatingWallVelocity;
origin (0 0 0);
axis (0 0 1);
omega constant 12.566370;
value uniform (0 0 0);

Impeller
type movingWallVelocity;
value uniform (0 0 0);

Inner region and arbitrary mesh interface (AMI patches)

`constant/dynamicMeshDict` – For sliding meshes

```
dynamicFvMesh        dynamicMotionSolverFvMesh;
motionSolverLibs     ( "libfvMotionSolvers.so" );
solver               solidBody;
cellZone             cell_inner_volume;
solidBodyMotionFunction  rotatingMotion;
origin    (0 0 0);
axis      (0 0 1);
omega constant 12.566370;
```

Motion libraries

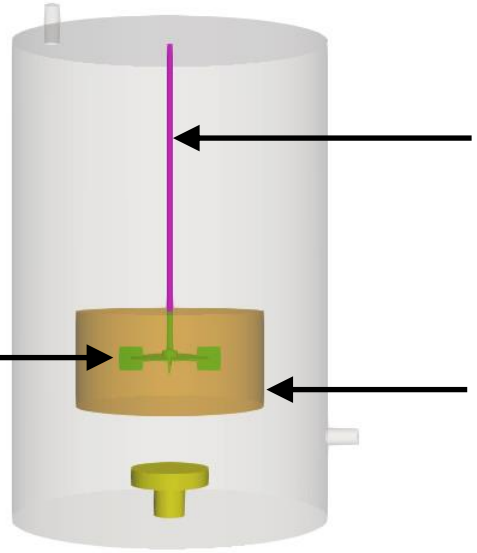
Solver for single body. For multiple bodies use the solver multiSolidBodyMotionSolver

Rotating zone selection

Motion type.
Omega is given in rad/s

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh



Shaft
type rotatingWallVelocity;
origin (0 0 0);
axis (0 0 1);
omega constant 12.566370;
value uniform (0 0 0);

Impeller
type movingWallVelocity;
value uniform (0 0 0);

Inner region and arbitrary mesh interface

constant/dynamicMeshDict – For sliding meshes

```
dynamicFvMesh    dynamicMotionSolverFvMesh;
motionSolverLibs ( "libfvMotionSolvers.so" );
solver           solidBody;
cellZone         cell_inner_volume;
solidBodyMotionFunction  rotatingMotion;
origin    (0 0 0);
axis      (0 0 1);
omega     constant 12.566370;
```

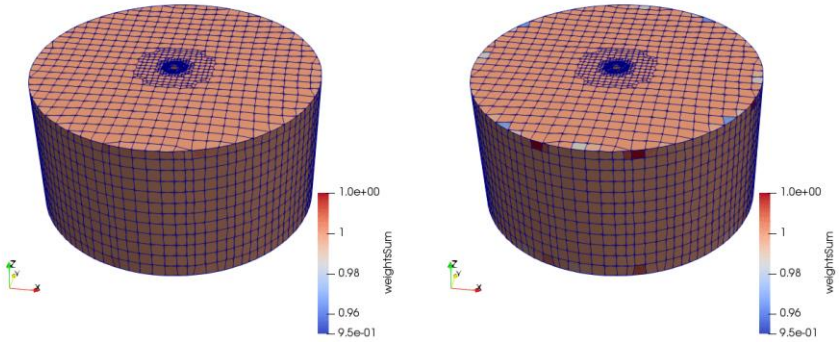
constant/MRFProperties – For MRF approach

```
cellZone    cell_inner_volume;
active      yes;

// Fixed patches (by default they move' with the MRF zone)
nonRotatingPatches ();

origin    (0 0 0);
axis      (0 0 1);
omega     constant 12.566370;
```

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor mesh

- The command `moveDynamicMesh -checkAMI` will print on screen the quality of the AMI interfaces for every time step.
- Ideally, you should get the AMI patches weights as close as possible to one.
- Weight values close to one will guarantee a good interpolation between the AMI patches.



http://www.wolfdynamics.com/training/movingbodies/image9.gif

...
...
...

Name of the AMI patch Name of the AMI patch

Calculating AMI weights between owner patch: AMI1 and neighbour patch: AMI2

AMI: Creating addressing and weights between 2476 source faces and 2476 target faces ← Number of faces in the AMI patches

AMI: Patch source sum(weights) min/max/average = 0.94746705, 1.0067199, 0.99994232 ← AMI1 patch weights

AMI: Patch target sum(weights) min/max/average = 0.94746692, 1.0004497, 0.99980782 ← AMI2 patch weights

...
...
...

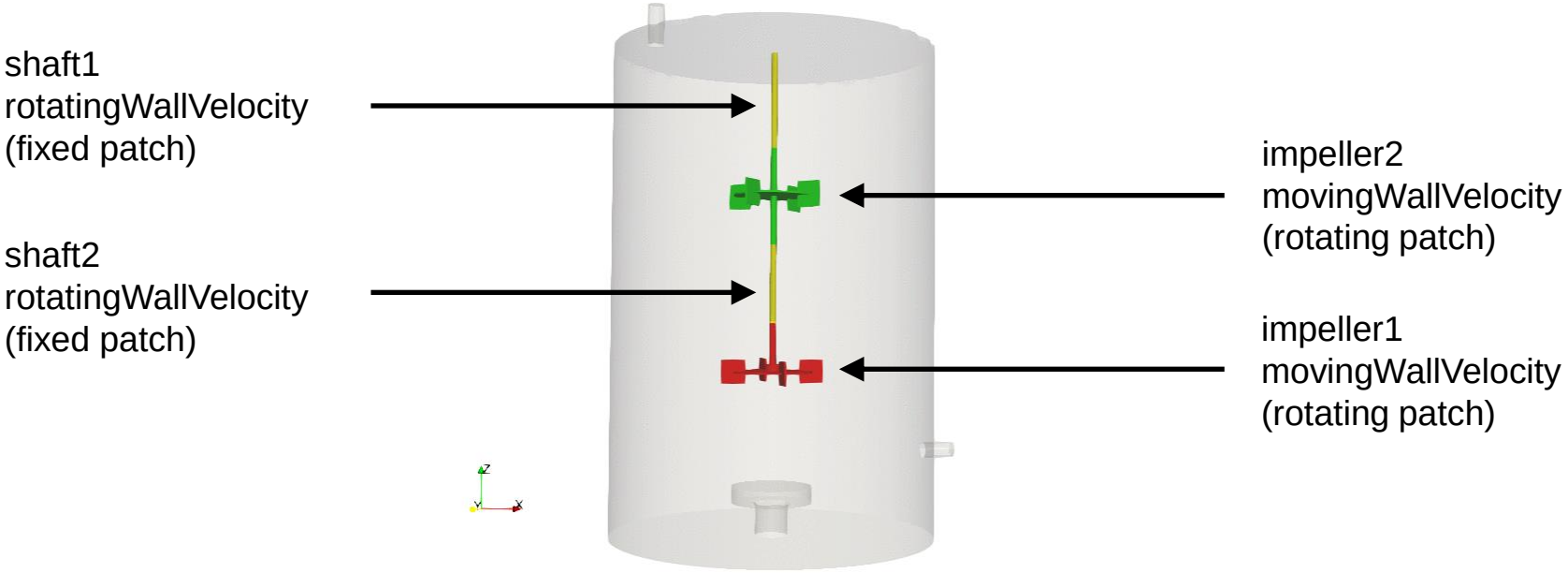
CSTR – Continuous stirring tank reactor mesh

- Additionally, to this case, in the directory **`$TM/sliding_MRF_meshes/CSTR/sliding_piso/`**, you will find additional cases dealing with MRF and importing a mesh from an external mesher.
  - **`./fluent_mesh`** – Importing a mesh from fluent
    - **`./case1`** – Zones/regions already defined
    - **`./case2`** – No zones/regions defined, it requires topological modifications
  - **`./MRF_piso`** – Unsteady MRF approach
  - **`./MRF_simple`** – Steady MRF approach
  - **`./sliding_piso`** – Sliding meshes approach

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor with two impellers

- And of course, it is possible to put several regions into motion (sliding meshes).
- Remember, for a single zone you can use the solver **solidBody**.
- For multiple regions, you will need to use the solver **multiSolidBodyMotionSolver**.
- You will need to create the **cellZones** and the **AMI** patches.
- The solver **multiSolidBodyMotionSolver** lets you use a single **cellZone** or multiple **cellZone**, so it is more general.



http://www.wolfdynamics.com/training/dynamicMeshes/sliding1.gif

Sliding meshes in OpenFOAM

CSTR – Continuous stirring tank reactor with two impellers

- For multiple regions, the dictionary *dynamicsMeshDict* looks like as follows:

```
dynamicFvMesh           dynamicMotionSolverFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solver                  multiSolidBodyMotionSolver;

multiSolidBodyMotionSolverCoeffs
{
        cell_inner_volume1
        {
                solidBodyMotionFunction  rotatingMotion;
                rotatingMotionCoeffs
                {
                        origin      (0 0 0);
                        axis        (0 0 1);
                        omega       constant 6.283185;  //360 deg/s
                }
        }

        cell_inner_volume2
        {
                solidBodyMotionFunction  rotatingMotion;
                rotatingMotionCoeffs
                {
                        origin      (0 0 0);
                        axis        (0 0 1);
                        omega       constant 6.283185;  //360 deg/s
                }
        }

}
```

Solver for multiple zones

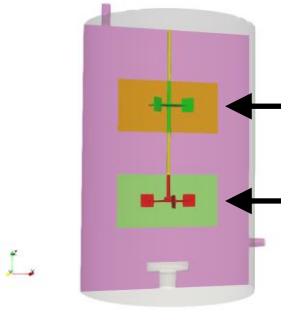
Zones entries

cellZone – created at meshing time

cellZone motion parameters

cellZone – created at meshing time

cellZone motion parameters



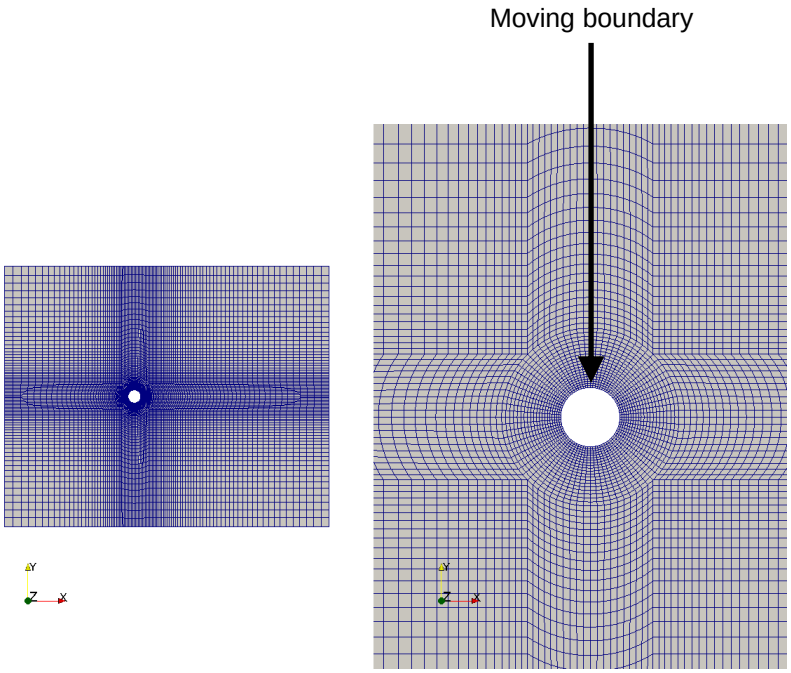
You will this tutorial in the directory
$TM/sliding_MRF_meshes/CSTR_twoImpellers

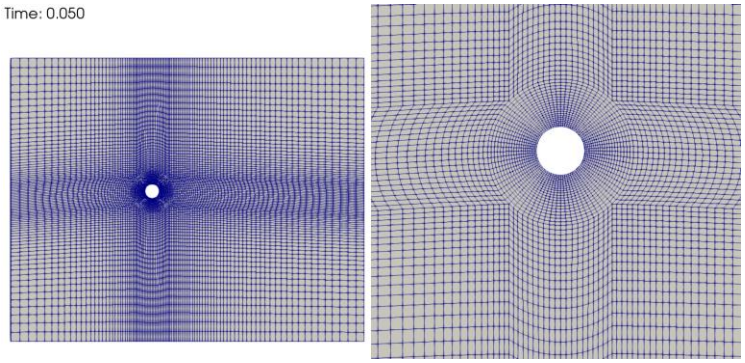
Roadmap

1. ~~Introduction – What are dynamic meshes?~~
2. ~~Adaptive mesh refinement in OpenFOAM~~
3. ~~Sliding meshes in OpenFOAM~~
4. **Morphing meshes in OpenFOAM**
5. Moving meshes in OpenFOAM
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

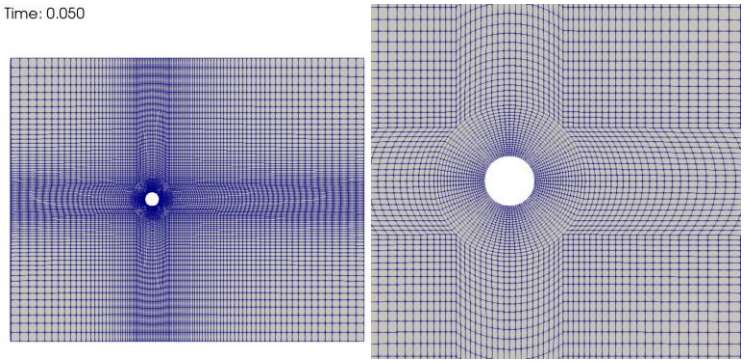
Morphing meshes in OpenFOAM

Oscillating cylinder – Prescribed motion





http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion3.gif

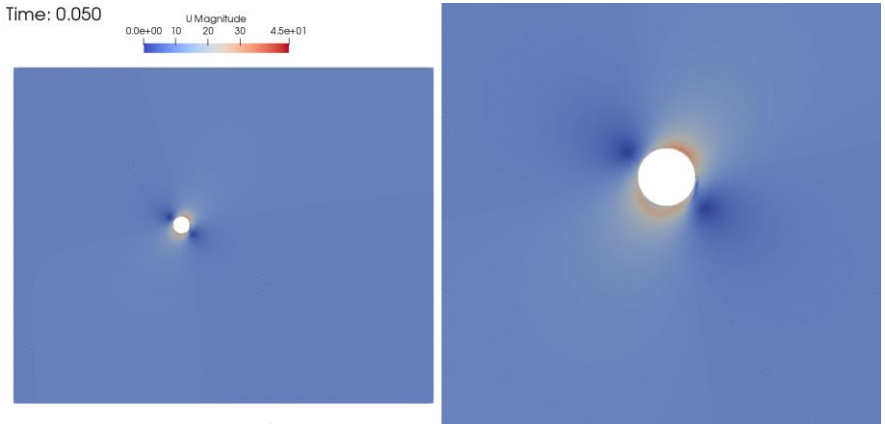


http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion4.gif

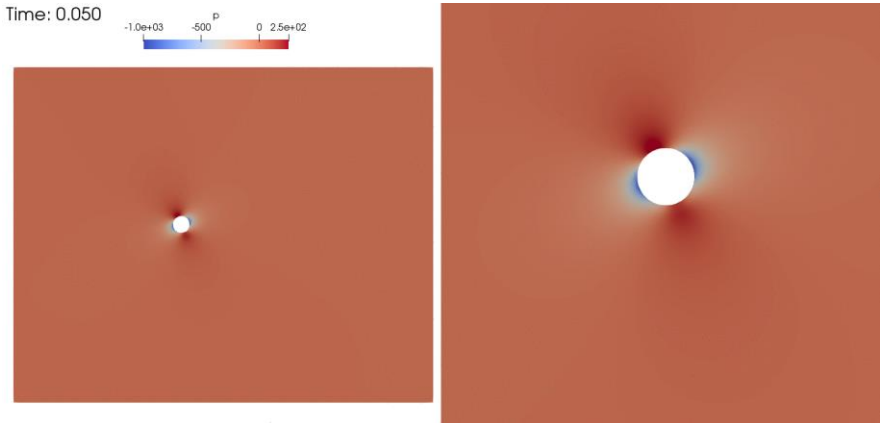
- The motion can be one of the predefined functions in OpenFOAM or an input table containing the body's linear displacement and angular displacement in function of time.
- Mesh morphing is also known as mesh diffusion and mesh smoothing technique.

Morphing meshes in OpenFOAM

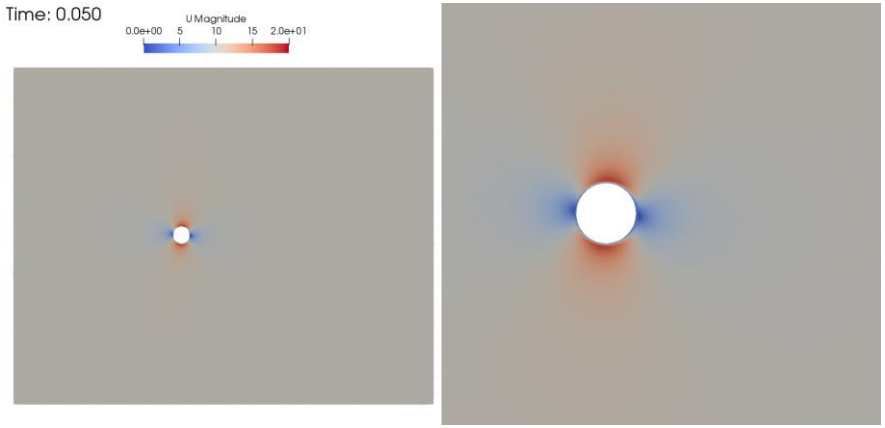
Oscillating cylinder – Prescribed motion



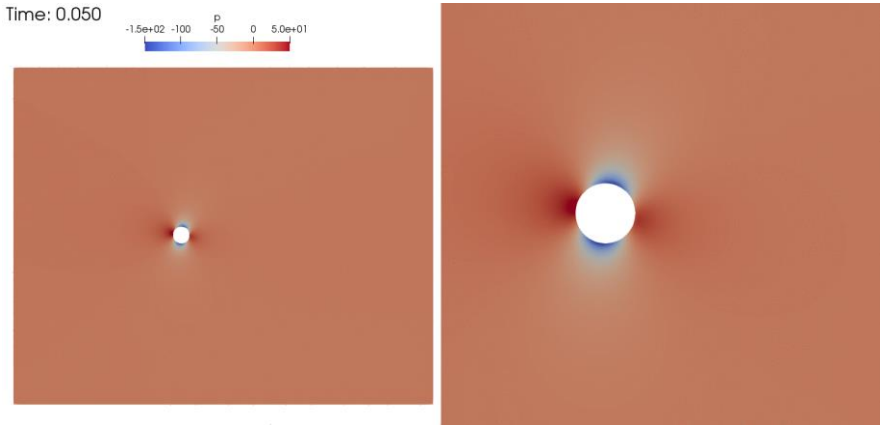
http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion5.gif



http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion6.gif



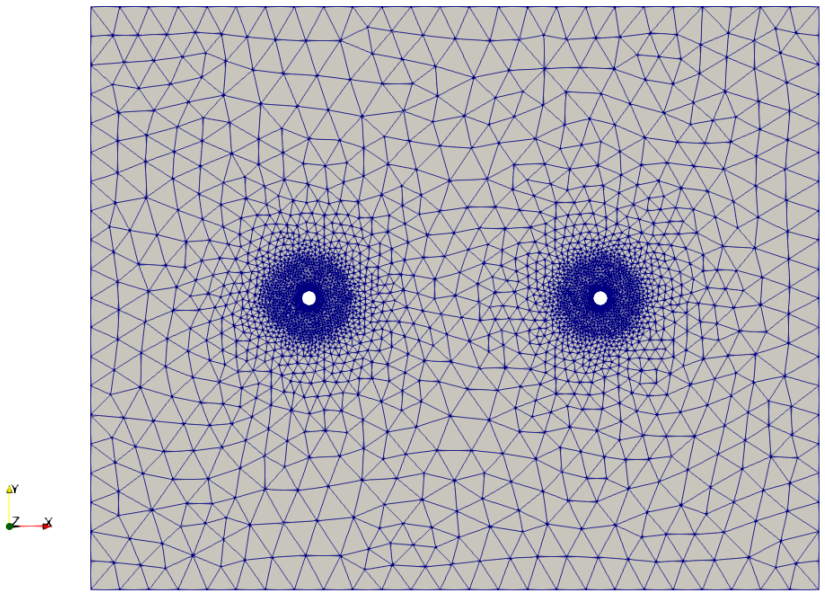
http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion7.gif



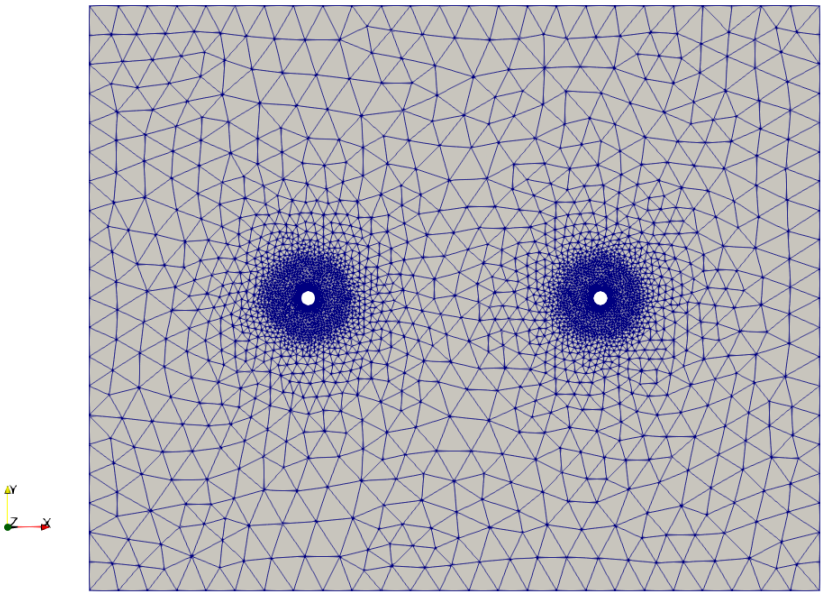
http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion8.gif

Morphing meshes in OpenFOAM

Oscillating cylinders – Prescribed motion with multiple bodies



http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion9.gif



http://www.wolfdynamics.com/training/dynamicMeshes/meshMotion10.gif

- With morphing meshes, large displacements with large time-steps (hence large CFL number), will likely result in low quality meshes or invalid meshes.
- The time-step must be chosen in such a way that it allows for a smooth mesh motion diffusion in the domain.
- When using mesh morphing, the solver will report the mesh quality.
- Remember to always check the mesh quality while the simulation is running, and it the case of low quality meshes, stop the simulation and fix the problems.

Morphing meshes in OpenFOAM

- Let us run the oscillating cylinder case.
- You will find this case in the directory:

`$TM/morphing_mesh/oscillatingCylinder/meshMotion`

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Morphing meshes in OpenFOAM

Oscillating cylinder – Prescribed motion

- In the dictionary *constant/dynamicMeshDict* we select the mesh morphing method and the boundary patch that it is moving.
- There are many mesh morphing methods implemented in OpenFOAM.
- Remember, to know all options available just misspelled something.
- Mesh morphing is based in diffusing or propagating the mesh deformation all over the domain.
- You will need to find the best method for your case.
- But in general, the setup used in this case works fine most of the times.

```
dynamicFvMesh      dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

solver             displacementLaplacian;

displacementLaplacianCoeffs
{
    diffusivity       inverseDistance (cylinder);
}
```

Mesh motion library

Motion library

Solver for mesh motion method. Based on solving the cell-centre Laplacian for the motion displacement. Many options available.

Method coefficients

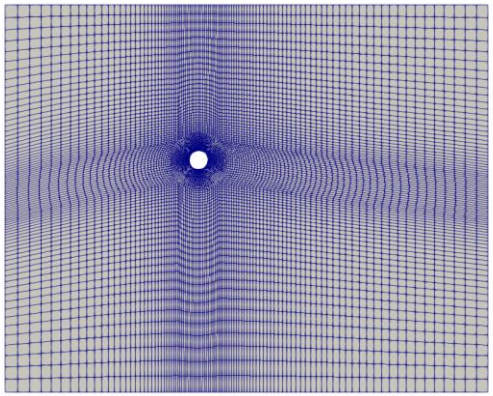
Mesh diffusion method – Many options available

Patch name – Can add multiple patches, e.g., (body1 body2)

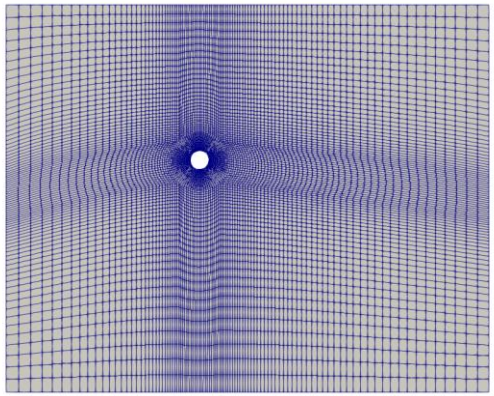
Morphing meshes in OpenFOAM

Oscillating cylinder – Prescribed motion

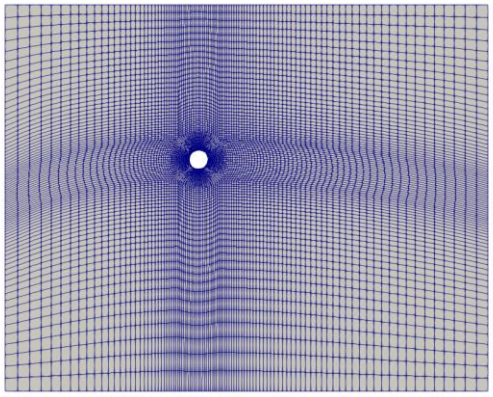
- Effect of different diffusivity methods (mesh morphing).



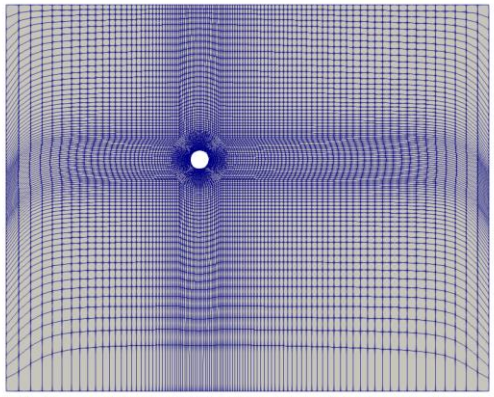
solver displacementLaplacian;
diffusivity inverseDistance (cylinder);



solver displacementLaplacian;
diffusivity inverseVolume (cylinder);



solver displacementLaplacian;
diffusivity quadratic inverseDistance (cylinder);



solver displacementLaplacian;
diffusivity exponential 0.6 inverseDistance (cylinder);

Oscillating cylinder – Prescribed motion

- In the dictionary *0/pointDisplacement* we select the prescribed body motion.
- In this case we are using **oscillatingDisplacement** for the **cylinder** patch.
- Each method has different input values. In this case it is required to define the amplitude and the angular velocity in rad/s.
- If the patch is not moving, we assign to it a **fixedValue** boundary conditions.

```
in
{
    type            fixedValue;
    value           uniform (0 0 0);
}

cylinder
{
    type            oscillatingDisplacement;

    amplitude       ( 0 1 0 );
    omega           6.28318;

    value           uniform ( 0 0 0 );
}
```

This is a fixed patch

Patch name – Different patches can have different prescribed motions

Built-in motion functions
Many options available.

Input values related to
motion assigned to the body

Dummy value for paraview

Oscillating cylinder – Prescribed motion

- If the patch is moving, we need to use the boundary condition **movingWallVelocity**.
- This is done in the dictionary *0/U*.

```
cylinder
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

- And as usual, you will need to adjust the numerics according to your physics.
- In this case we need to solve the new fields **cellDisplacement** and **diffusivity**, which are related to the mesh motion and morphing.
- In the dictionary *fvSolution*, you will need to add a linear solver for the field **cellDisplacement**.
- In the dictionary *fvSchemes*, you will need to add the discretization schemes related to the mesh morphing diffusion method **laplacian(diffusivity, cellDisplacement)**. You can use the default discretization method (*e.g.*, Gauss linear limited 1.0).

Oscillating cylinder – Prescribed motion

- At this point, we are ready to run the simulation.
- You will find the instructions of how to run the cases in the file *README.FIRST* located in the case directory.
- Before running the simulation, you can check the mesh motion.
- During this check, you can use large time-steps as we re not computing the solution, we are only interested in checking the motion.
- To check the mesh motion, type in the terminal:

```
1.  $> moveDynamicMesh -noFunctionObjects
```

- We leave to the reader to play around with the different options for mesh motion method, mesh diffusion method, and prescribed mesh motion.

Solver types:
displacementComponentLaplacian
displacementInterpolation
displacementLaplacian
displacementLayeredMotion
displacementLinearMotion
displacementSBRStress
multiSolidBodyMotionSolver
solidBody
velocityComponentLaplacian
velocityLaplacian

Mesh diffusivity methods:
directional
exponential
file
inverseDistance
inverseFaceDistance
inversePointDistance
inverseVolume
motionDirectional
quadratic
uniform

Patches prescribed motions:
angularOscillatingDisplacement
angularOscillatingVelocity
oscillatingDisplacement
oscillatingVelocity
solidBodyMotionDisplacement
surfaceDisplacement
surfaceSlipDisplacement
timeVaryingMappedFixedValue
uniformInterpolatedDisplacement
waveDisplacement

Oscillating cylinder – Prescribed motion

- In the directory **`$TM/morphing_mesh/oscillatingCylinder/`**, you will find additional cases dealing with moving bodies and mesh morphing.
  - **`./solution1`** – Moving body using a predefined displacement function.
  - **`./solution2`** – Moving body using a tabular input.
    - The tabular input reads linear displacement and angular rotation in function time. During the overset meshes tutorials we will address this input format.
    - Tabular inputs are the most flexible way to use complex motions.
    - To use tabular inputs, we must use the **solidBodyMotionDisplacement** prescribe patch motion boundary condition.
    - This boundary condition permits the user assign a fixed value boundary condition using the solid body motion functions, that is, **axisRotationMotion**, **linearMotion**, **multiMotion**, **oscillatingLinearMotion**, **oscillatingRotatingMotion**, **rotatingMotion**, **SDA**, **solidBodyMotionFunction**, **tabulated6DoFMotion**.

Morphing meshes in OpenFOAM

What about if my mesh is static?

- If a solver supporting dynamic meshes (*e.g.*, `pimpleFoam`, `interFoam`, `rhoPimpleFoam`) finds the dictionary file *`constant/dynamicMeshDict`*, it will parse it and use the dynamic meshes definition declared.
- If your mesh is static, you can erase the dictionary file *`constant/dynamicMeshDict`*, and the solver will assume that the mesh is neither moving nor changing.
- Alternatively, you can leave the dictionary file *`constant/dynamicMeshDict`*, and modify it as follows, so your mesh is static,

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh      staticFvMesh;   ←  Library for static meshes


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

Morphing meshes in OpenFOAM

- Let us run the falling-floating body case – Rigid body motion
- You will find this case in the directory:

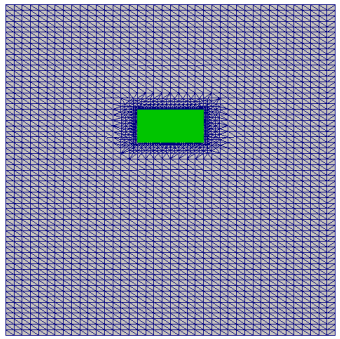
$TM/morphing_mesh/fallingObject_remeshSHM/mesh1

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

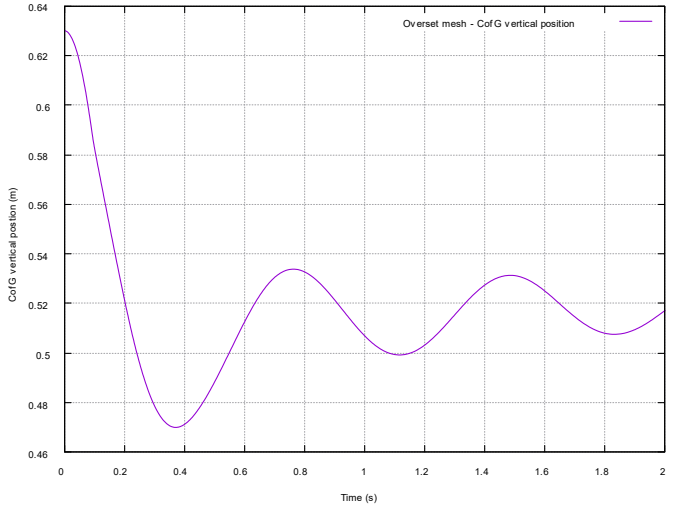
Morphing meshes in OpenFOAM

Falling-floating body – Rigid body motion

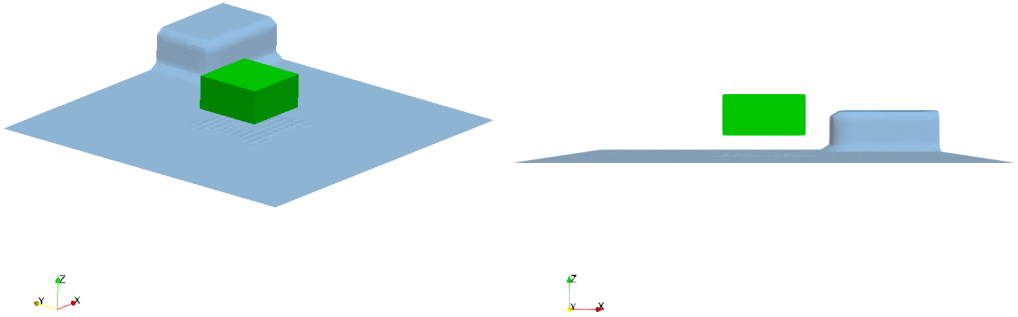
Time: 0.000000



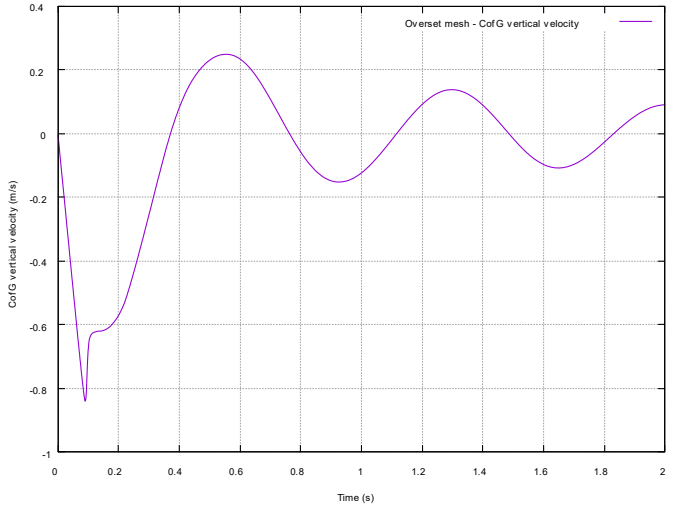
http://www.wolfdynamics.com/training/dynamicMeshes/dof1.gif



Time: 0.000000

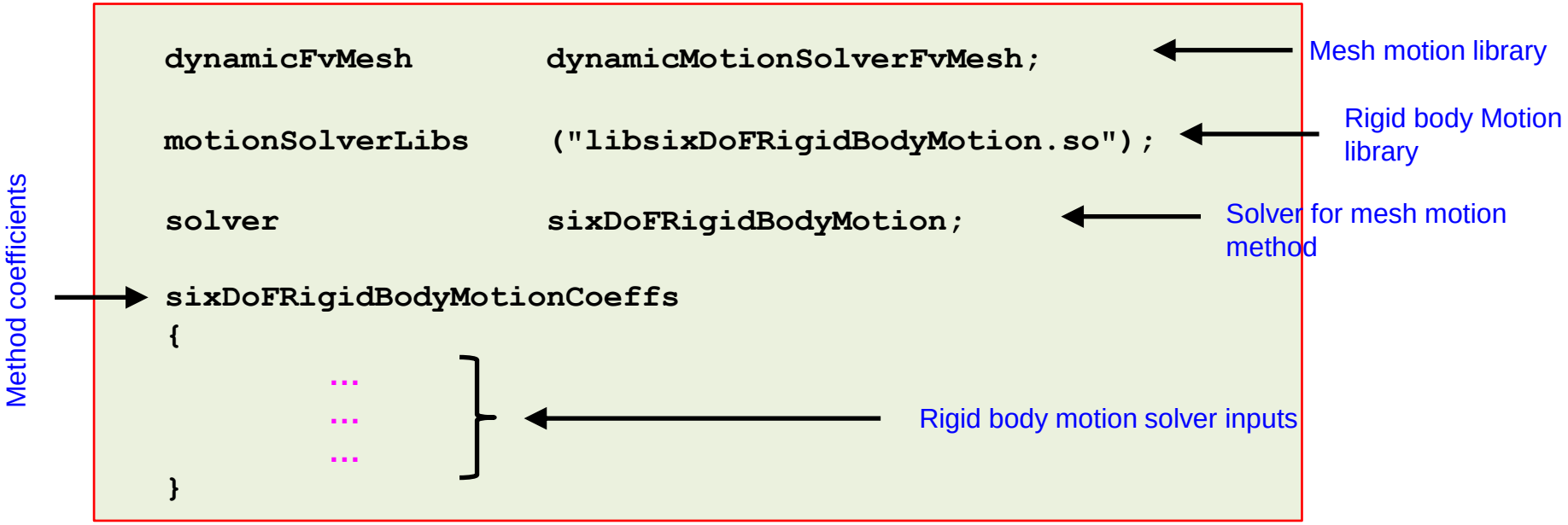


http://www.wolfdynamics.com/training/dynamicMeshes/dof2.gif



Falling-floating body – Rigid body motion

- As for prescribed motion, in rigid body motion the mesh morphing is based in diffusing or propagating the mesh deformation all over the domain.
- In the dictionary *constant/dynamicMeshDict* we select the mesh morphing library and rigid body motion library.
- The rigid motion solver will compute the response of the body to external forces.
- In the dictionary *constant/dynamicMeshDict* we define all the inputs required by the rigid motion solver.
- In this case we are using the dynamic motion library **sixDoFRigidBodyMotion**, this library works with a single body.
- To work with multiple bodies, you will need to use the library **rigidBodyMotion.** We will use this library during the overset tutorials.



Morphing meshes in OpenFOAM

Falling-floating body – Rigid body motion

- The dictionary *constant/dynamicMeshDict* (continuation).

```
sixDoFRigidBodyMotionCoeffs
{
    patches          (floatingObject);

    innerDistance    0.1;
    outerDistance    0.4;

    //velocity           (0 0 1)

    centreOfMass     (0.5 0.5 0.63);
    mass             3;
    momentOfInertia  (0.08 0.08 0.1);

    report           on;

    ...
    ...
```

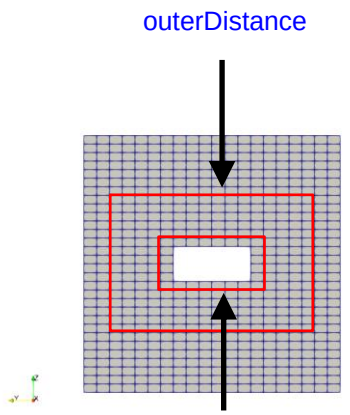
Moving patch

Mesh deformation limits.
The mesh will not be deformed in the fringe located within innerDistance and outerDistance (distance normal to the wall)

This define the initial velocity of the body the default value is (0 0 0).
This entry is optional

Physical properties of the body

Report on screen position of the body



innerDistance
Set it to zero if you do not want to apply mesh morphing to the inner region

Falling-floating body – Rigid body motion

- The dictionary *constant/dynamicMeshDict* (continuation).

```
sixDoFRigidBodyMotionCoeffs
{
    ...
    ...
    ...

    accelerationRelaxation 0.7;

    accelerationDamping 1.0;

    solver
    {
        type Newmark;
    }

    ...
    ...
    ...
```

$$\phi_p^n = d_{factor} \times (\phi_p^{n-1} + \alpha(\phi_p^{n^*} - \phi_p^{n-1}))$$

Relaxation factor used to stabilize the rigid body motion solver.
For no relaxation set it to 1. Typical values are between 0.9-0.3

Damping factor used only if you are interested in reaching an equilibrium (trim) condition.
For no damping set it to 1.0.
Use with caution it will severely damp the forces acting on the body.

Rigid body motion solver used to solve the ODE governing the motion of the body.
Many options available.

Falling-floating body – Rigid body motion

- The dictionary *constant/dynamicMeshDict* (continuation).

```
constraints
{
    fixedAxis
    {
        sixDoFRigidBodyMotionConstraint axis;
        axis (0 1 0);
    }

    fixedLine
    {
        sixDoFRigidBodyMotionConstraint line;
        centreOfRotation (0.5 0.5 0.5);
        direction (0 0 1);
    }

}

restraints
{

}

}
```

Motion constraints
If you do not give any constraint, the body is free to move in all directions.

If you assign a constraint, the body is free to move in the specified direction

Body restraints
Restraints can be used to damp the acceleration of the body.

In this case, we are not using restraints

Falling-floating body – Rigid body motion

- In the dictionary *0/pointDisplacement* we select the body motion.
- For rigid body motion, the body motion is computed by the solver, therefore, we use the boundary condition calculated.

```
floatingObject
{
    type            calculated;
    value           uniform (0 0 0);
}
```

- And as the patch is moving, we need to use the boundary condition **movingWallVelocity**.
- This is done in the dictionary *0/U*.

```
floatingObject
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

Falling-floating body – Rigid body motion

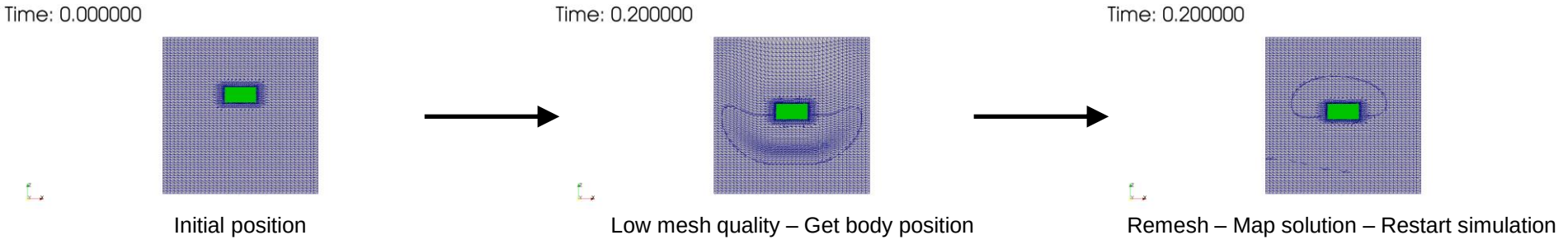
- And as usual, you will need to adjust the numerics according to your physics.
- In the case directory, you will also find the script *extractData*. This script can be used to extract the position of the body during the simulation.
- In order to use the *extractData* script, you will need to save the log file of the simulation.
- At this point, we are ready to run the simulation.
- Remember, in the case directory you will find the scripts *run_mesh.sh*, *run_solver.sh* and *run_all.sh*, you can use these scripts to run the case automatically. To run all the steps, type in the terminal:
    - `$> sh run_all.sh`

- You will find the instructions of how to run the cases in the file *README.FIRST* located in the case directory.

Morphing meshes in OpenFOAM

Falling-floating body – Rigid body motion

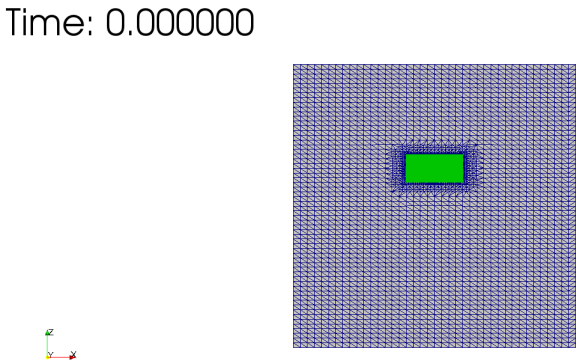
- It is also possible to use mesh morphing with remeshing.
- That is, when the quality of the mesh is too low, you can stop the simulation, get the position of the body, remesh the domain, map the solution and keep computing.
- And it is recommended to reduce the number of interpolated solutions, so the errors introduced when restarting the simulation are reduced.



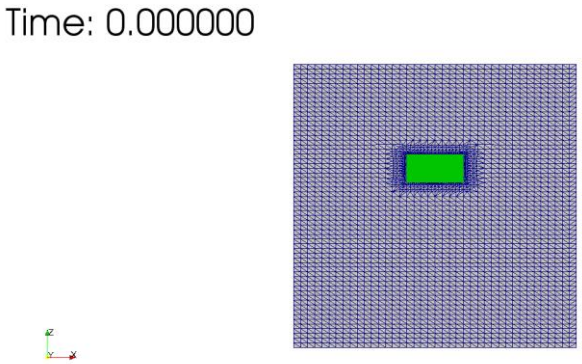
Initial position

Low mesh quality – Get body position

Remesh – Map solution – Restart simulation



http://www.wolfdynamics.com/training/dynamicMeshes/dof1.gif

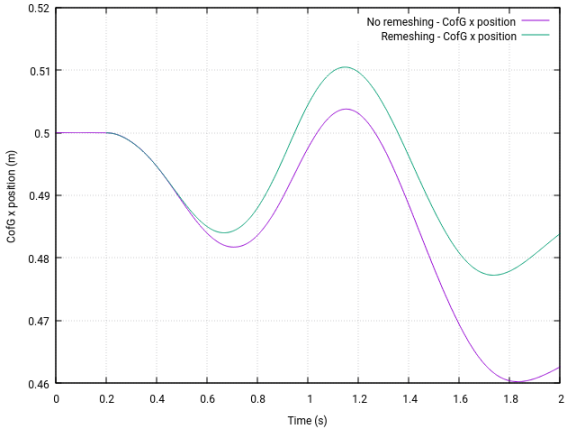


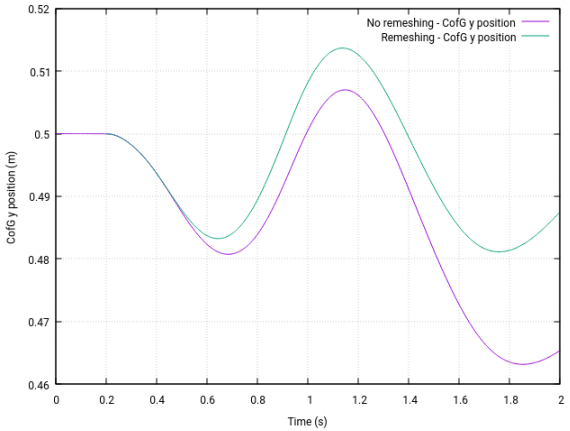
http://www.wolfdynamics.com/training/dynamicMeshes/dof3.gif

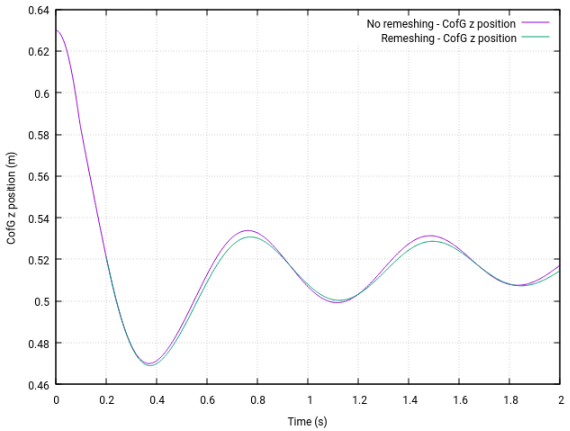
Morphing meshes in OpenFOAM

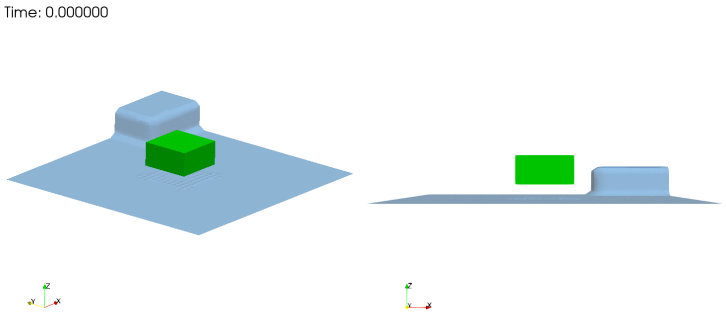
Falling-floating body – Rigid body motion

- This is maybe the best approach to deal with large deformations, but it requires careful parametrization and case setup.
- You will find this case ready to run in the directory:
  `$TM/morphing_mesh/fallingObject_remeshSHM/mesh2`

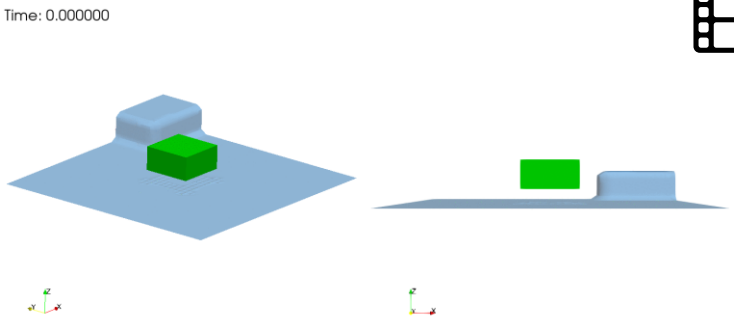








No remesh
http://www.wolfdynamics.com/training/dynamicMeshes/dof2.gif



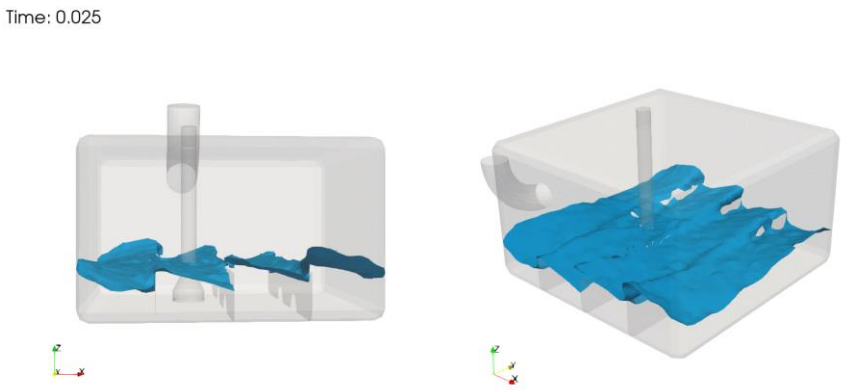
Remesh
http://www.wolfdynamics.com/training/dynamicMeshes/dof4.gif

Roadmap

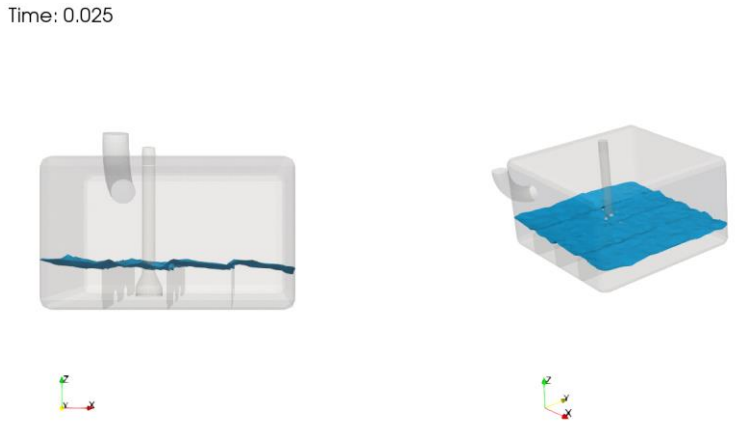
1. ~~Introduction – What are dynamic meshes?~~
2. ~~Adaptive mesh refinement in OpenFOAM~~
3. ~~Sliding meshes in OpenFOAM~~
4. ~~Morphing meshes in OpenFOAM~~
5. **Moving meshes in OpenFOAM**
6. Overset meshes in OpenFOAM
7. Final remarks – General guidelines

Moving meshes in OpenFOAM

Moving meshes/domain – Sloshing tank



http://www.wolfdynamics.com/training/dynamicMeshes/sloshing1.gif



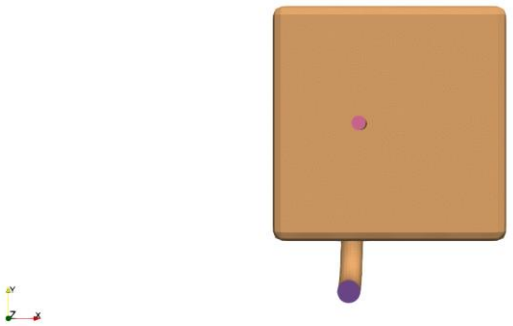
http://www.wolfdynamics.com/training/dynamicMeshes/sloshing2.gif

- In this kind of simulations, we aim at moving the whole domain.
- These simulations are particularly useful when dealing with sloshing tanks cases.
- As for prescribed motion, we need to assign the solid body motion to the whole domain.
- As usual, in the dictionary `constant/dynamicMeshDict` we select the solid body motion library and prescribed motion.

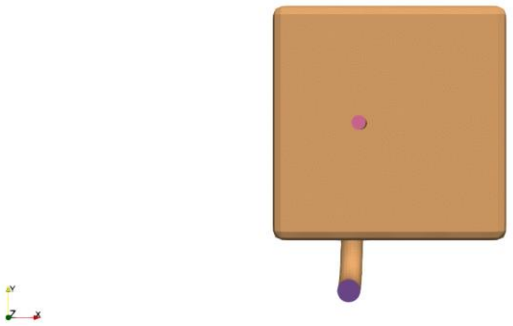
Moving meshes in OpenFOAM

Moving meshes/domain – Sloshing tank

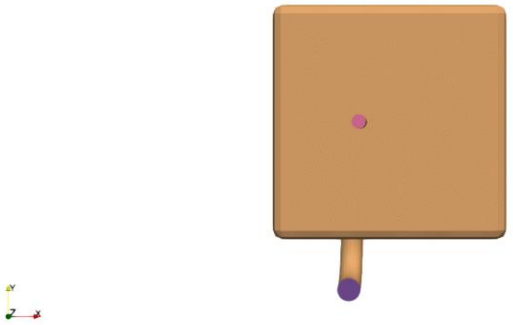
Solid body – Different prescribed motions



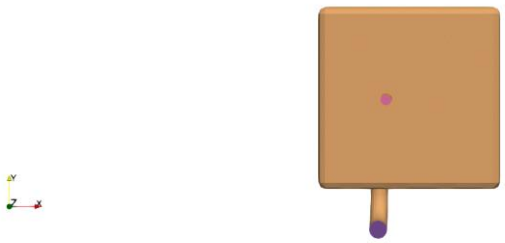
Oscillating rotating
http://www.wolfdynamics.com/training/dynamicMeshes/motion1.gif



Oscillating linear
http://www.wolfdynamics.com/training/dynamicMeshes/motion2.gif



Combination of oscillating rotating and oscillating linear
http://www.wolfdynamics.com/training/dynamicMeshes/motion3.gif



Arbitrary motion using a tabular input
http://www.wolfdynamics.com/training/dynamicMeshes/motion4.gif

Moving meshes in OpenFOAM

- Let us run the sloshing tank case.
- You will find this case in the directory:

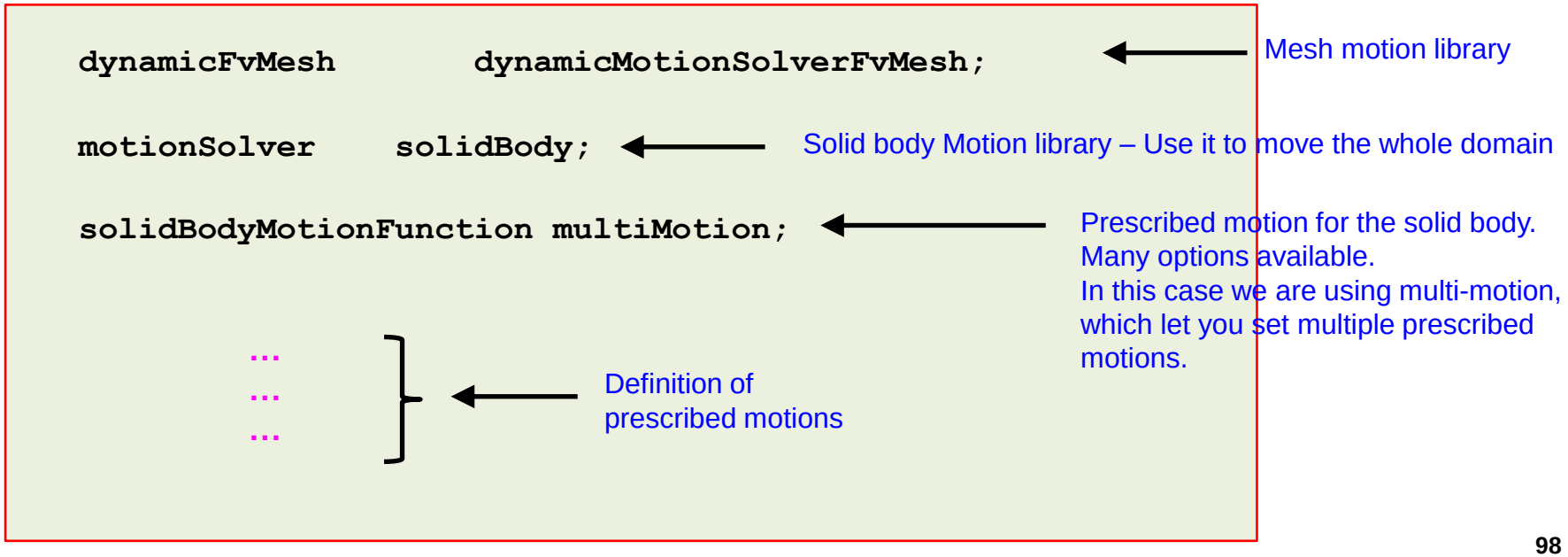
```
$TM/moving_mesh/sloshing_tank_baffles/baffles_multimotion/
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Moving meshes in OpenFOAM

Sloshing tank

- Contrary to morphing meshes, in these simulations we do not morph the mesh, we only move the domain (similar to what we did in sliding meshes).
- In the dictionary *constant/dynamicMeshDict* we select the mesh motion libraries and prescribed motions.
- In the **solidBodyMotionFunction** entry you can define a singe motion. But using **multiMotion** is more general as it let you choose multiple prescribed motions.
- Remember, to know all options available just misspelled something.



Moving meshes in OpenFOAM

Sloshing tank

- Definition of multiple prescribed motions using the **multiMotion** solid body function.

```
motion1
{
    solidBodyMotionFunction oscillatingRotatingMotion;
    oscillatingRotatingMotionCoeffs
    {
        origin          (0.25 0.25 0.25);
        amplitude       (0 0 60);
        omega           3.14159;
    }
}

motion2
{
    solidBodyMotionFunction oscillatingLinearMotion;
    oscillatingLinearMotionCoeffs
    {
        amplitude      (0.25 0 0);
        omega          3.14159;
    }
}
```

User given name (used to enumerate the prescribed motions)

Prescribed motion

Input values related to motion assigned to the body. Amplitude in degrees and omega in rad/s.

User given name (used to enumerate the prescribed motions)

Prescribed motion

Input values related to motion assigned to the body. Amplitude in meters and omega in rad/s.

Sloshing tank

- As all the walls are moving, we need to use the boundary condition **movingWallVelocity**.
- This is done in the dictionary *0/U*.

```
cylinder
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

- And as usual, remember to adjust the numerics according to your physics.
- At this point, we are ready to run the simulation.
- Remember, in the case directory you will find the scripts *run_mesh.sh*, *run_solver.sh* and *run_all.sh*, you can use these scripts to run the case automatically. To run all the steps, type in the terminal:
    - `$> sh run_all.sh`
- To check the mesh motion, type in the terminal:
    - `$> moveDynamicMesh -noFunctionObjects`
- You will find the instructions of how to run the cases in the file *README.FIRST* located in the case directory.

Sloshing tank

- At this point, we are ready to run the simulation.
- You will find the instructions of how to run the cases in the file *README.FIRST* located in the case directory.
- Before running the simulation, you can check the mesh motion.
- During this check, you can use large time-steps as we re not computing the solution, we are only interested in checking the motion.
- To check the mesh motion, type in the terminal:

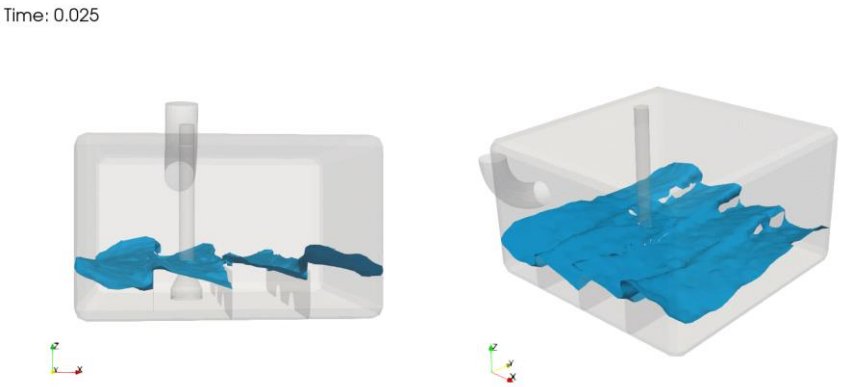
```
1.  $> moveDynamicMesh -noFunctionObjects
```

- We leave to the reader to play around with the different options for **solidBodyMotionFunction**.

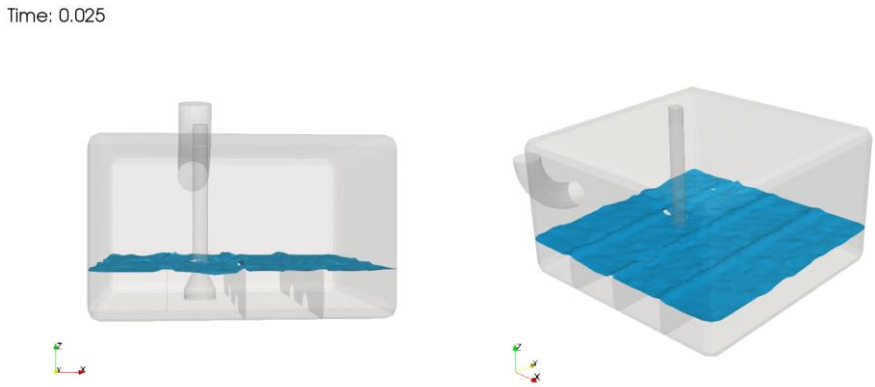
solidBodyMotionFunction motion types:
axisRotationMotion
linearMotion
multiMotion
oscillatingLinearMotion
oscillatingRotatingMotion
rotatingMotion
SDA
solidBodyMotionFunction
tabulated6DoFMotion

Sloshing tank

- Remember, when bodies/patches are moving you need to assign to all the moving walls the **movingWallVelocity** boundary condition.
- This boundary condition will add the mesh velocity.
- In this example, you can clearly see the difference in the solution when we use and when we do not use the boundary condition **movingWallVelocity.**
- As you can see, if we do not use the boundary condition **movingWallVelocity** for all moving walls the solution does not look that good.



Moving walls with **movingWallVelocity** boundary condition
http://www.wolfdynamics.com/training/dynamicMeshes/sloshing1.gif



Moving walls with **fixedvalue** boundary condition
http://www.wolfdynamics.com/training/dynamicMeshes/sloshing3_badbc.gif

Roadmap

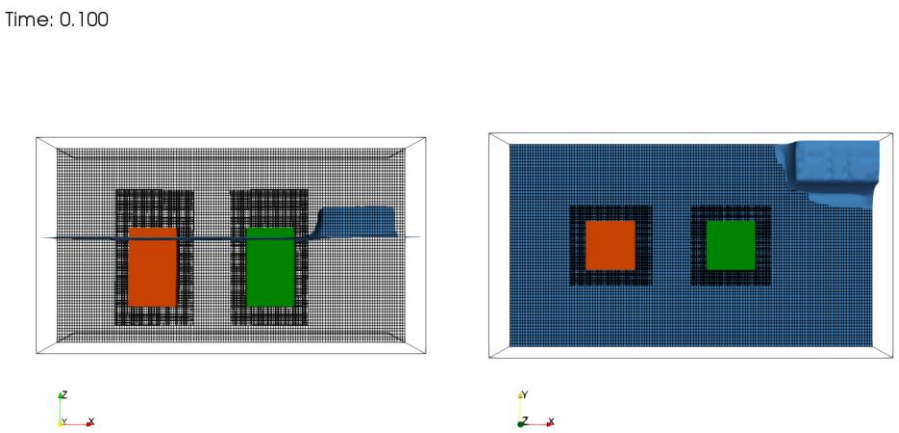
1. ~~Introduction – What are dynamic meshes?~~
2. ~~Adaptive mesh refinement in OpenFOAM~~
3. ~~Sliding meshes in OpenFOAM~~
4. ~~Morphing meshes in OpenFOAM~~
5. ~~Moving meshes in OpenFOAM~~
6. **Overset meshes in OpenFOAM**
7. Final remarks – General guidelines

A few preliminary remarks about overset meshes

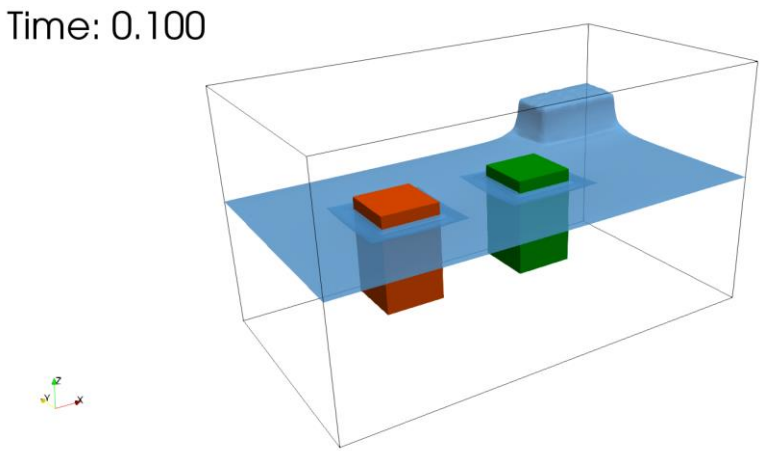
- By using overset meshes, simulations involving complex motion (prescribed, 6DOF, or FSI) of single or multiple bodies that were extremely difficult or impossible to simulate using traditional moving meshes methods (mesh morphing, layering, or remeshing), are now tractable.
- If you are working with unstructured meshes and there are no moving bodies, it makes no sense paying the extra computational cost inherent to overset meshes.
- Have in mind that overset meshes can add numerical diffusion to the solution, not to mention that the interpolation is non-conservative.
- Do not take overset meshes as a silver bullet. Simulations using overset meshes require careful planning and expertise.

Overset mesh example in OpenFOAM

Rigid body motion with multiple bodies and VOF



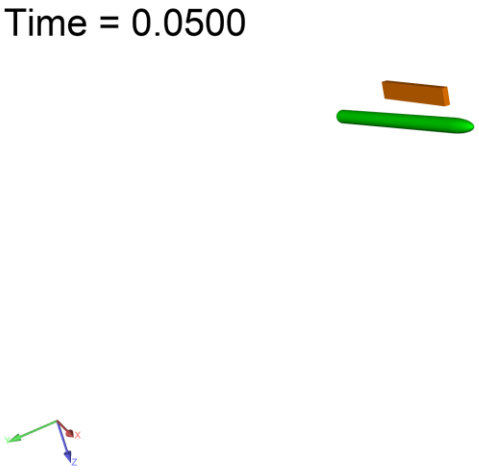
http://www.wolfdynamics.com/training/dynamicMeshes/floating_overset1.gif



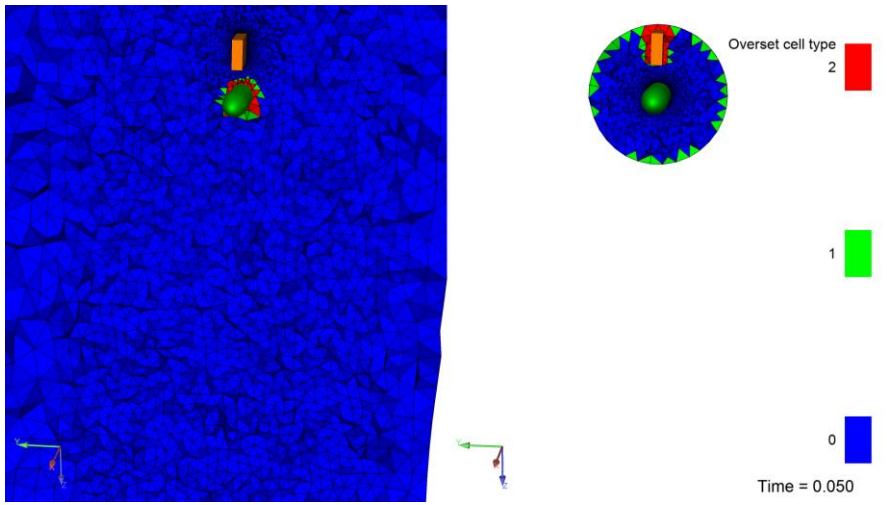
http://www.wolfdynamics.com/training/dynamicMeshes/floating_overset2.gif

Overset mesh example in OpenFOAM

Store separation – Rigid body motion



http://www.wolfdynamics.com//wiki/of_conf2019/f15.gif

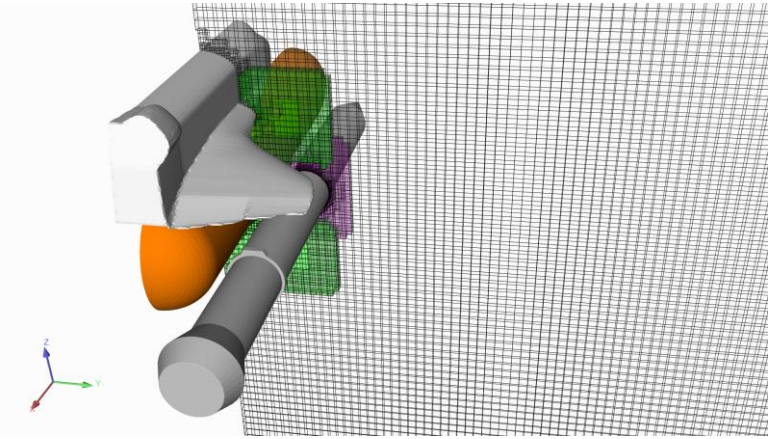


http://www.wolfdynamics.com//wiki/of_conf2019/f17.gif

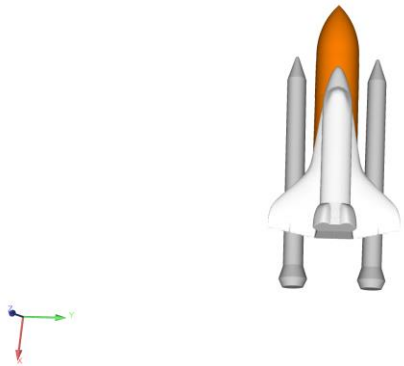
Overset meshes in OpenFOAM

Overset mesh example in OpenFOAM

Space shuttle – Booster release – Prescribed motion

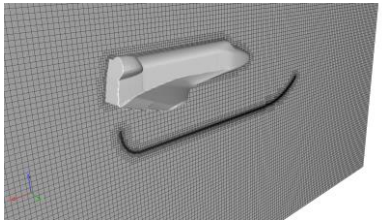


http://www.wolfdynamics.com//wiki/of_conf2019/f19.gif

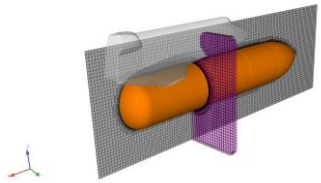


http://www.wolfdynamics.com//wiki/of_conf2019/f20.gif

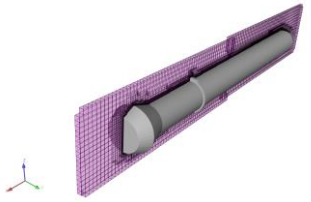
Component mesh 1



Component mesh 2

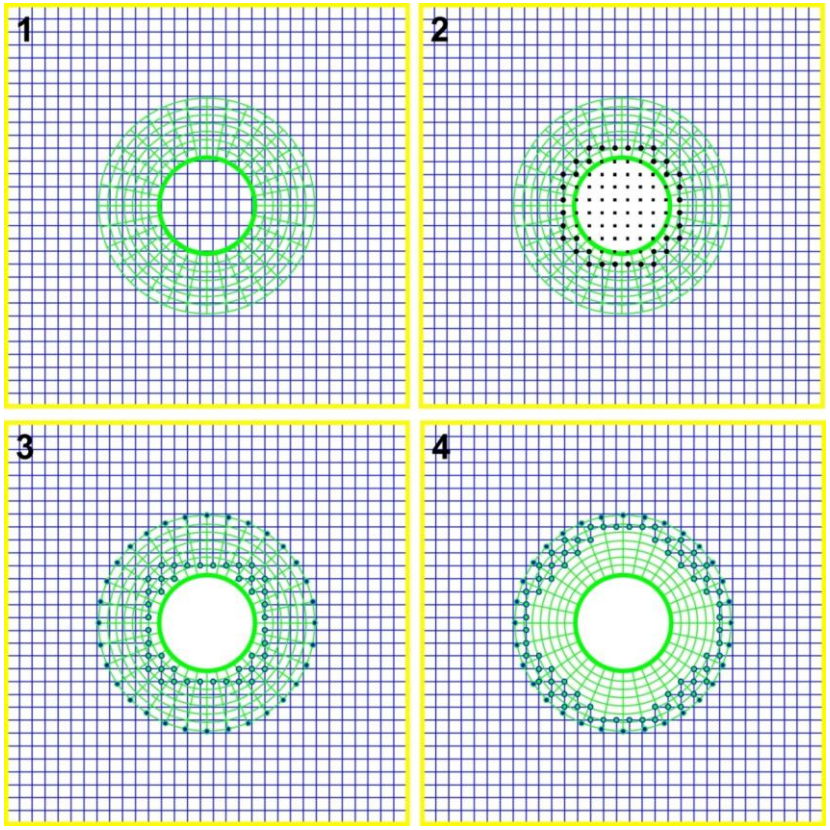


Component mesh 3



Overview of overset meshes – Development timeline

- The overset meshes (**OM**) method consists in generating a set of component meshes (**CM**) that cover the domain and overlap where they meet.
- Domain connectivity between the **CM** is obtained through proper interpolation in the overlapping areas.
- The **CM** can be structured or unstructured.
- In the CFD community, the **OM** method has been in use since the early 1980's.
- It was then, and it is now recognized as an attractive approach for treating problems with moving bodies and complex geometries (think structured meshes/solvers).
- **OM** are also known as overlapping grids, overset composite grids, composite overlapping meshes, chimera meshes, patches grids, composite grids.

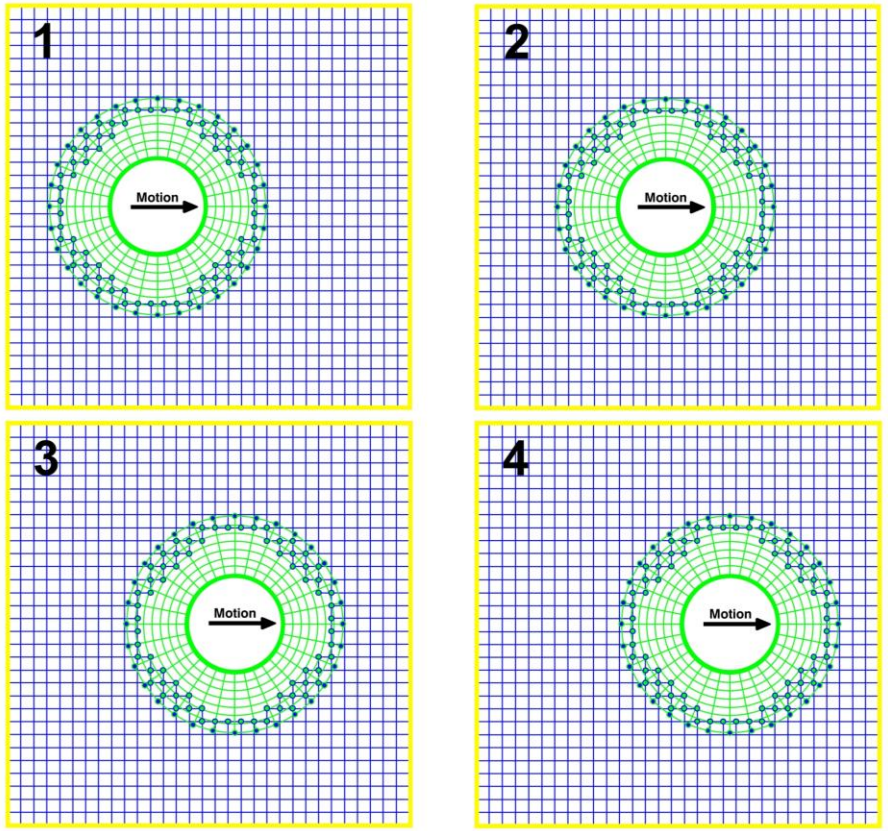


1. Component meshes (**CM**) – The **CM** are generated separately.
2. Hole cutting – Identification of unused points.
3. Identification of valid interpolation points (this is a valid mesh).
4. Optimized overset mesh – Mesh set with the minimum overlap region.

Overset meshes in OpenFOAM

Overview of overset meshes – Development timeline

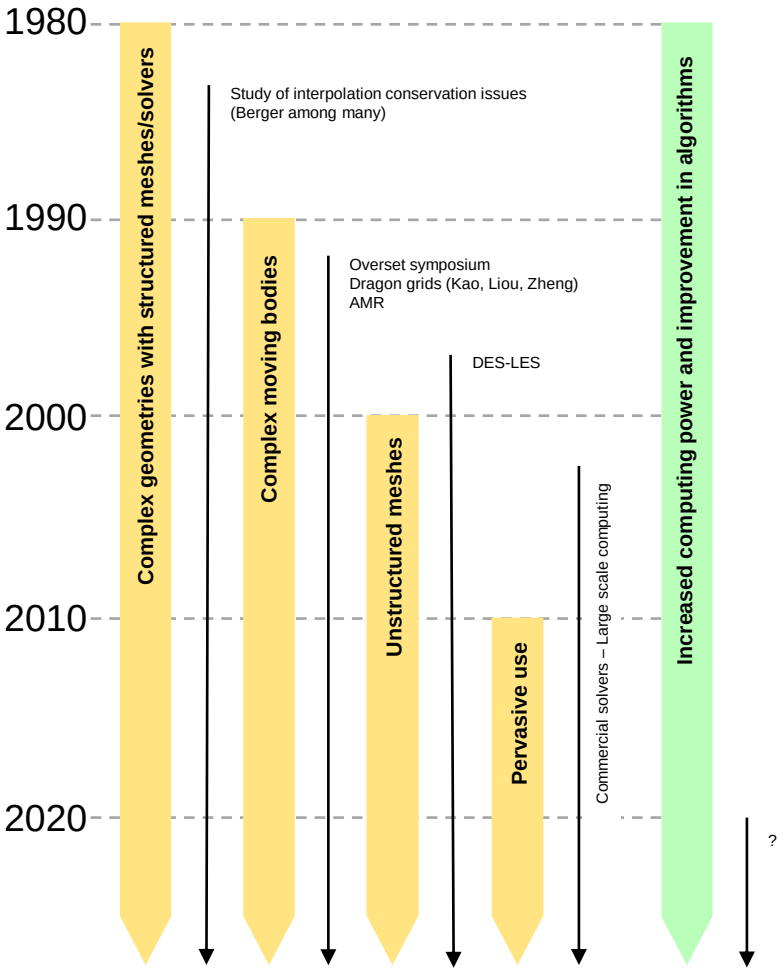
- If the **CM** are moving, overset connectivity information, such as interpolation stencils and unused points regions (Chimera holes), is recomputed each time-step.
- The motion of the **CM** may be a user defined function, may obey the Newton-Euler equations for the case of rigid body motion or may be the boundary nodes displacement in response to the stresses exerted by the fluid pressure for the case of FSI problems.
- **OM** can easily handle multiple bodies undergoing relative motion.
- They can even handle collisions.
- Overset meshes guarantees high quality meshes even for very large displacements.



- Moving overset mesh.
- The interpolation stencil and Chimera holes are recomputed every time-step.
- The illustrated overset mesh corresponds to a mesh set with the minimum overlap between component meshes.
- But sets with larger overlap regions can be used as well.

Overset meshes in OpenFOAM

Overview of overset meshes – Development timeline

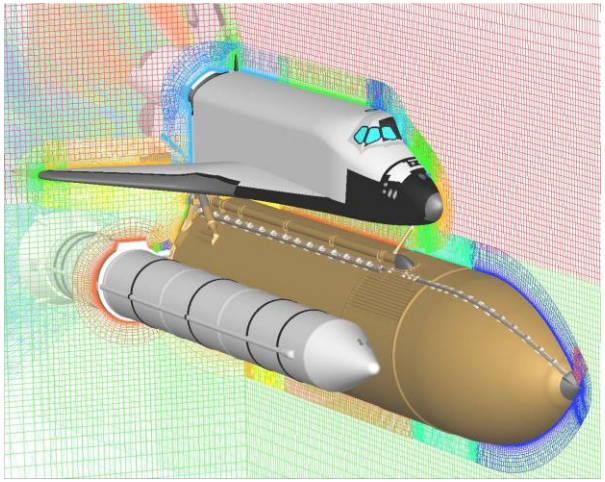


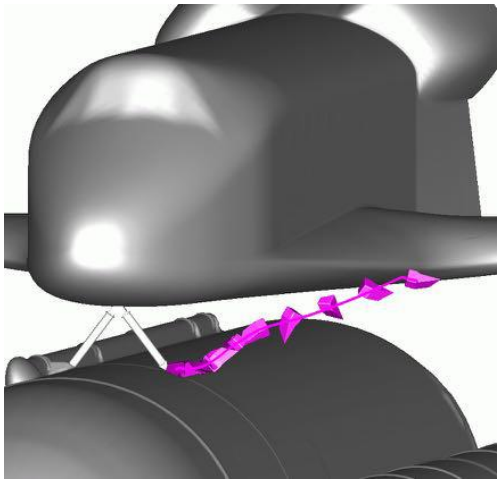
- Maybe the first use of overlapping grids was reported by Volkov in the late 1960's.
- The method was further developed and promoted by Starius and Kreiss in the late 1970's.
- It was formally introduced into the CFD community in the early 1980's by the pioneering work of Benek, Buning, Dougherty, Meakin, Steger, Suhs.
- Since the 1990's it has been heavily used to deal with complex geometries and moving bodies (Benek, Boger, Bunning, Chan, Chesshire, Dougherty, Gomez, Henshaw, Meakin, Noack, Petersson, Rogers, Steger, Suhs, among many).
- Since 2000's, the use of overset meshes with unstructured meshes gained popularity.
- From 2010's most commercial CFD solvers and many open-source simulation frameworks use overset meshes.
- Symposium on Overset Composite Grids and Solution Technology (http://oversetgridsymposium.org/).
  - Biyearly event.
  - First edition took place in 1992 – NASA Ames Research Center, California.
  - Next edition: 2020 – NASA Langley, Virginia.

Overset meshes in OpenFOAM

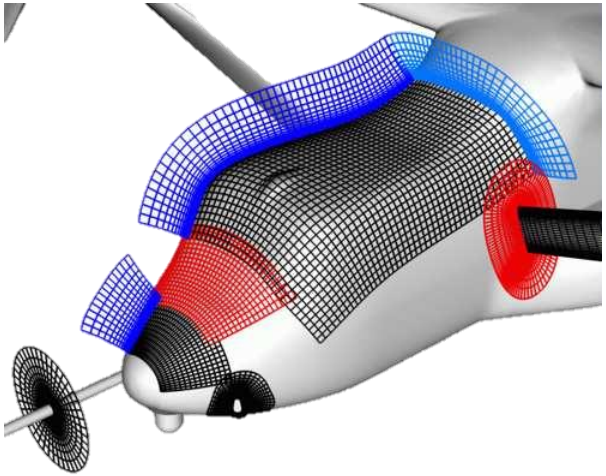
Overview of overset meshes – Development timeline

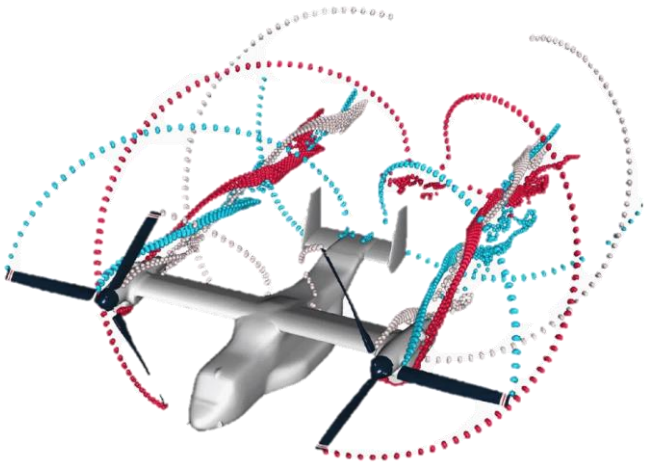
- Overset meshes are used to solve the most challenging moving bodies problems.





Space shuttle
Figure credit: P. Buning, W. Chan, R. Gomez, S. Pandya.
Copyright on the images is held by the contributors. Apart from Fair Use, permission must be sought for any other purpose.





V-22 Osprey
Figure credit: W. Chan, R. Meakin, W. Wissink.
Copyright on the images is held by the contributors. Apart from Fair Use, permission must be sought for any other purpose.

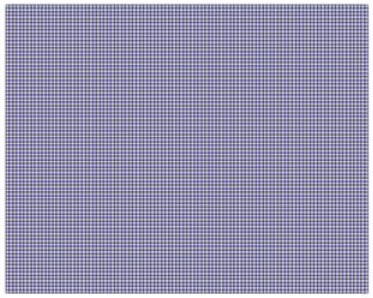
The grammar of overset meshes in OpenFOAM

- The process of assembling overset meshes in OpenFOAM is very straightforward.
- Four basic steps are involved:
  1. Generate component meshes and merge them together (done by the user).
  2. Define overset patches (done by the user).
  3. Assign zones (done by the user).
  4. Compute stencils and assign cell type (done by the overset library).

- These steps are common for every CFD solver that uses overset meshes.
- The difference is the tools and methods used to merge meshes, assign zones, define grid priorities, compute stencils, and diagnosing the overset assembly.
- Let us illustrated these steps using an overset set with three component meshes. For this, we will use the classical cylinder case (Re = 200).

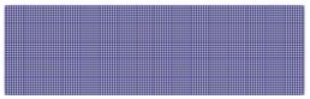
The grammar of overset meshes in OpenFOAM

- Step 1 → Generate component meshes and merge them together (done by the user).



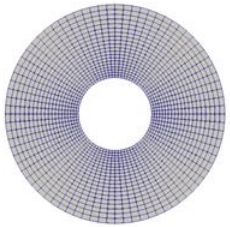
Component mesh 1 → **all**

+



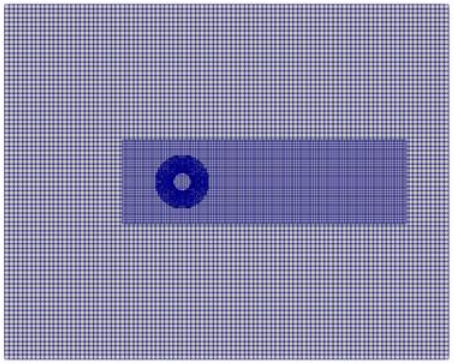
Component mesh 2 → **refinementZone**

+

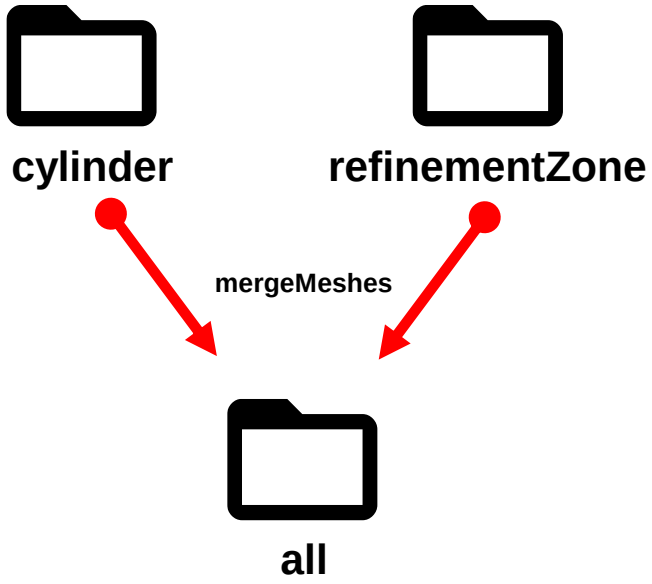


Component mesh 3 → **cylinder**

=



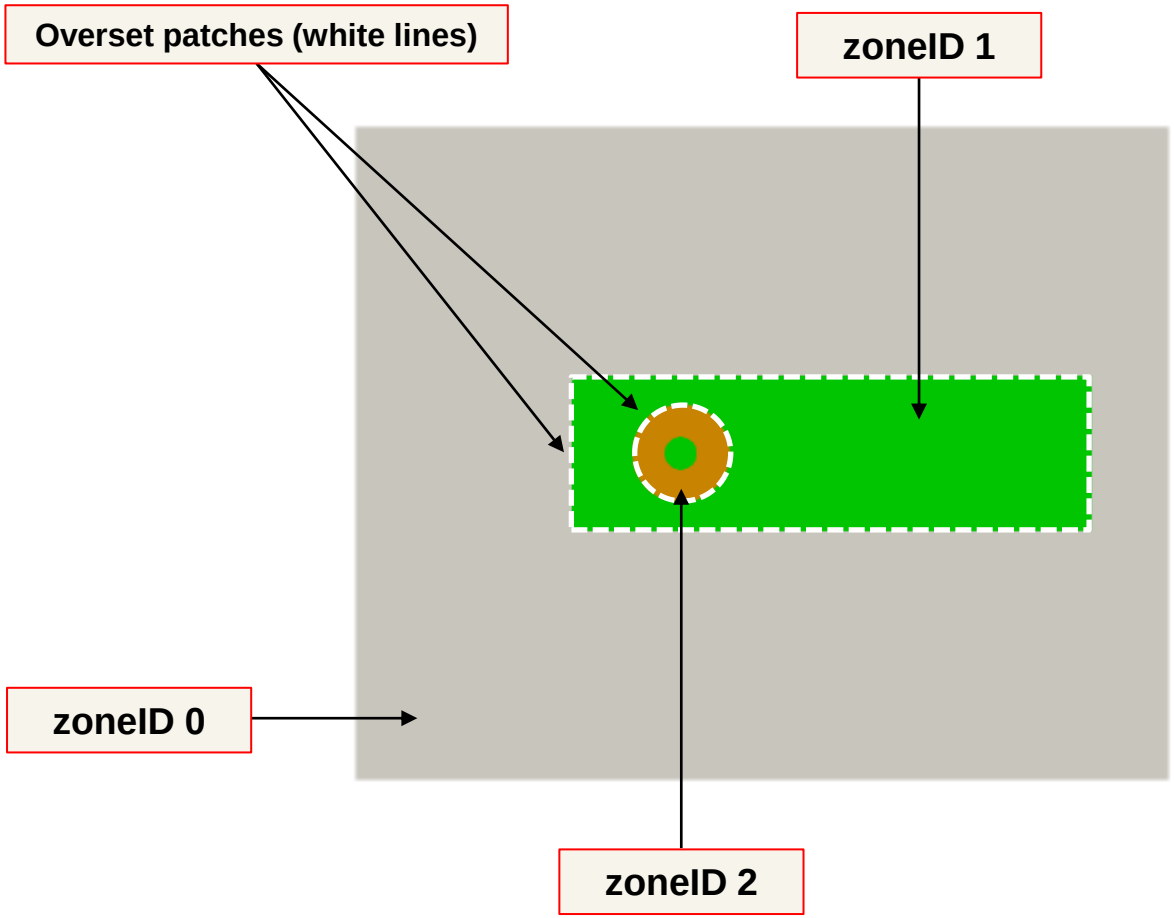
- The cell type can be any of the cells supported by OpenFOAM.
- The meshes can be 2D and 3D.
- The meshes can be generated using any meshing utility (OpenFOAM or third-party library).



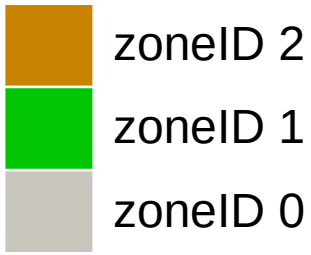
- Each **CM** is considered an individual case; therefore, they are generated in different directories.
- To assemble an overset mesh, you need to generate each **CM** in separated directories.
- Then, you merge them together using the utility **mergeMeshes**.
- You merge the meshes in a single directory. In this case, the component meshes **cylinder** and **refinementZone** are merged in the directory **all**.
- Notice that the directory **all** also contains a mesh (background mesh in this case).

The grammar of overset meshes in OpenFOAM

- Step 2 → Define overset patches (done by the user).

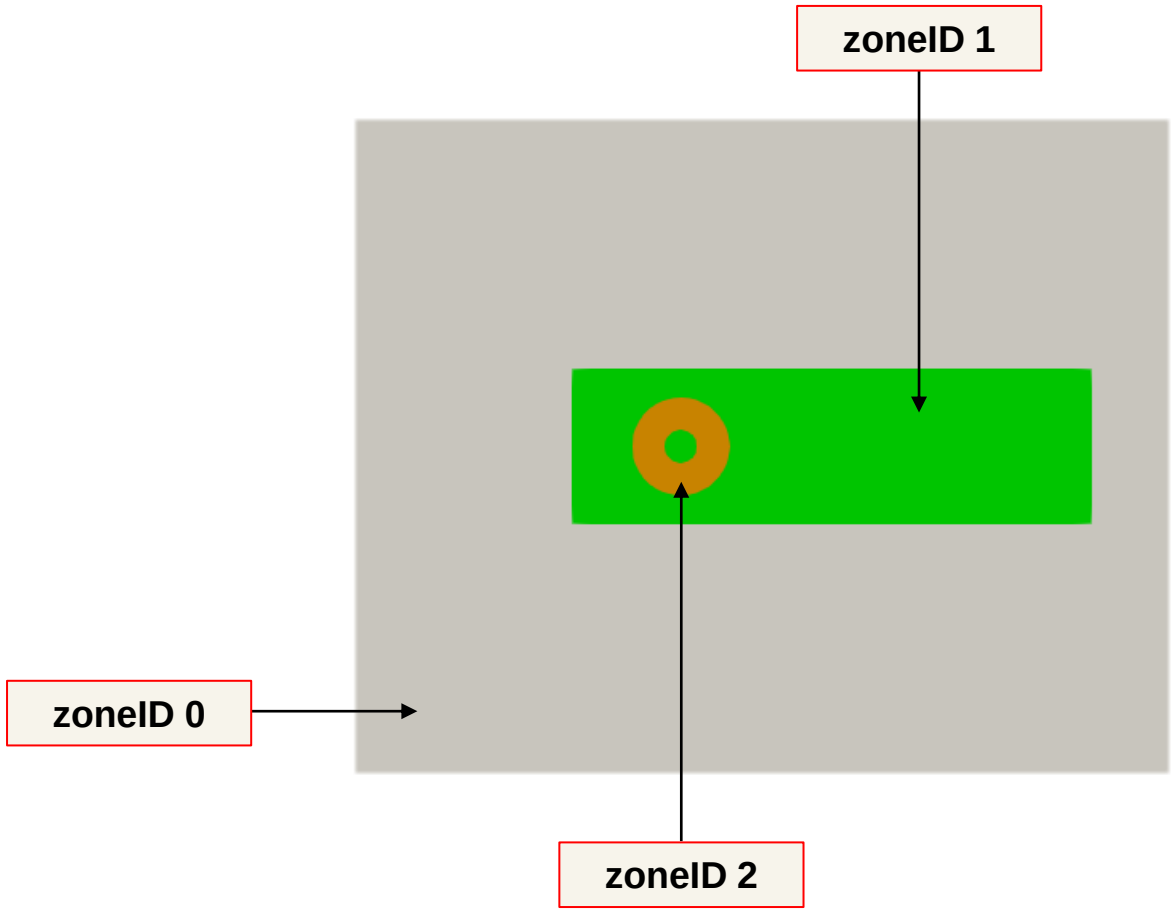


- The overset patches are defined by the user.
- They have the same name (defined by the user when generating the **CM**).
- And they are grouped together automatically when merging meshes.
- Overset patches can intersect each other.
- They can also intersect other patches (walls).
- However, walls cannot intersect other walls (no collisions) or go out of the domain (escape).

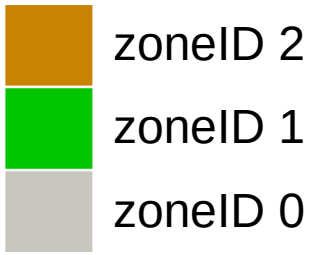


The grammar of overset meshes in OpenFOAM

- Step 3 → Assign zones (done by the user).

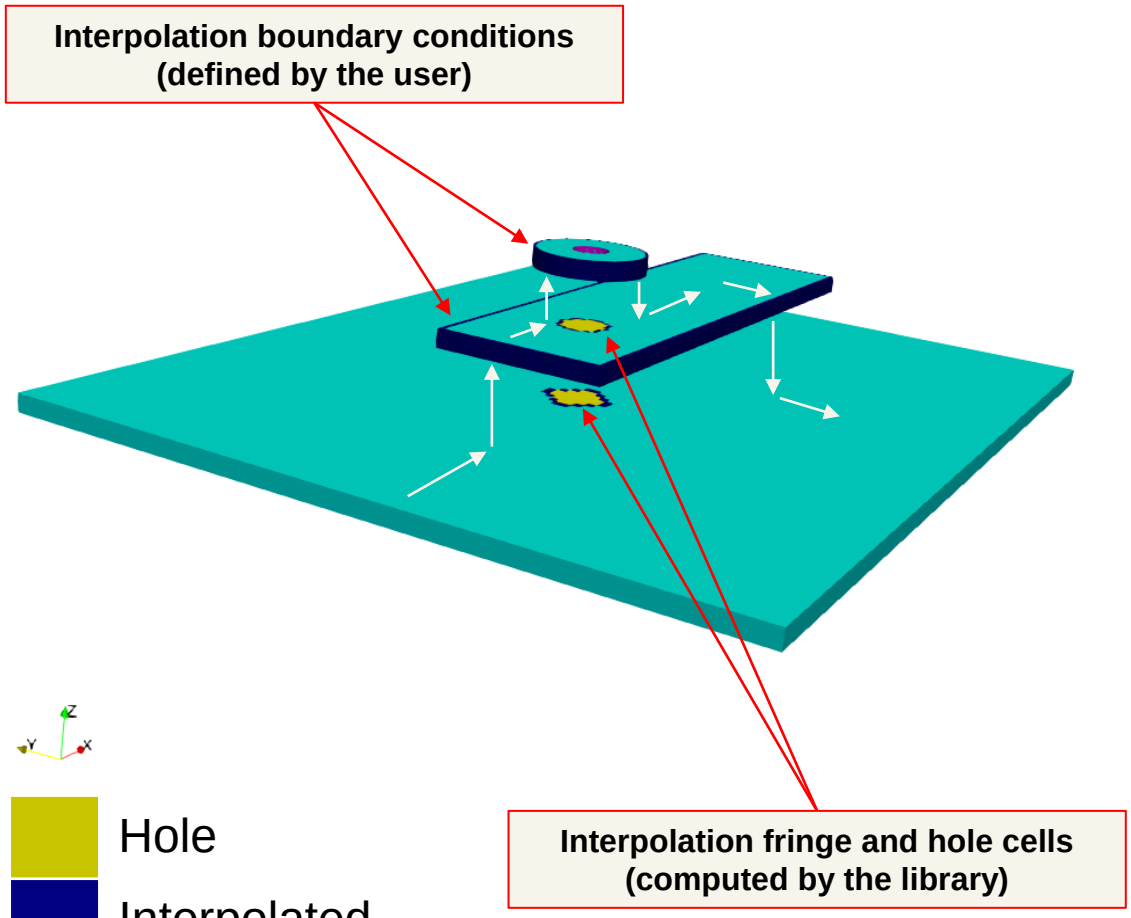


- A zone identification index (**zoneID**) is assigned to each component mesh after they have been merged.
- It is recommended to assign **zoneID** 0 to the background mesh (**all** in this case).
- The background mesh usually is the mesh that is not moving, the mesh with inlet and outlet patches, or the mesh that does not have overset patches.
- The **zoneID** index is used to establish the grid priorities.



The grammar of overset meshes in OpenFOAM

- Step 4 → Compute stencils and assign cell type (done by the overset library).

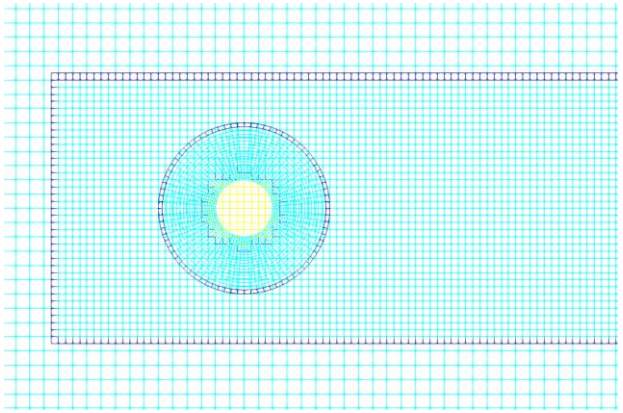


- The **overset patches** of each **CM** are defined by the user.
- The **interpolation fringe** close to the walls and the **hole cells** are computed automatically by the overset library.
- The cell types are defined as follows: **hole cells** (the solution is not computed), **interpolated cells** (the solution is interpolated from mesh-to-mesh), and **calculated cells** (the solution is computed).
- The **interpolated cells** can be classified as **acceptors** (receive information) and **donors** (send information).
  - The **donor** cells can be **interpolated** or **calculated** cells.

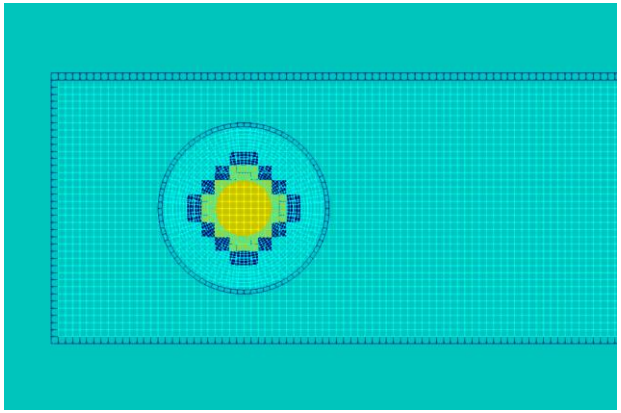
Cell type	Cell type index
Hole	2
Interpolated	1
Calculated	0

The grammar of overset meshes in OpenFOAM

- Step 4 → Compute stencils and assign cell type (done by the overset library).

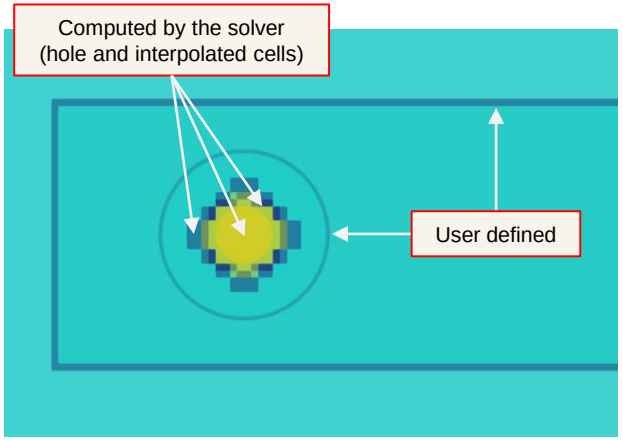


Wireframe visualization – All **CMs**

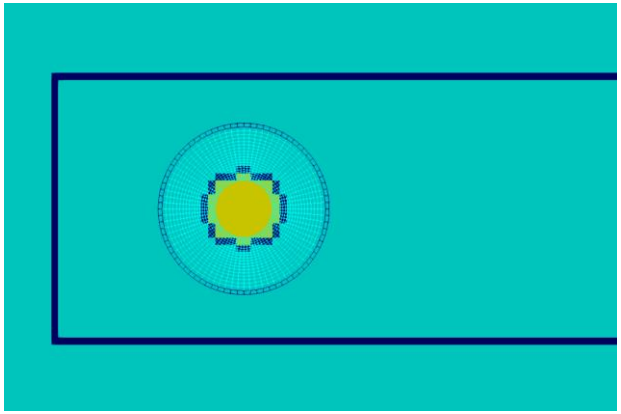


Wireframe visualization (**refinementZone** and **cylinder CM**)

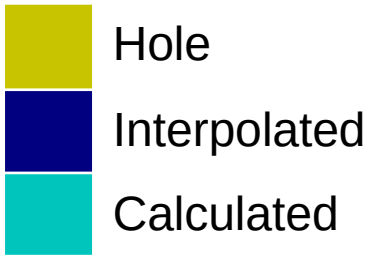
- Cells dimension close to interpolated cells should be of the same size to minimize interpolation errors.
- When computing the solution, an overset interpolation method must be chosen.
- Options available:
  - **cellVolumeWeight**
  - **inverseDistance**
  - **trackingInverseDistance**
  - **leastSquares** (recommended by us)



Contour visualization with transparency – All **CMs**

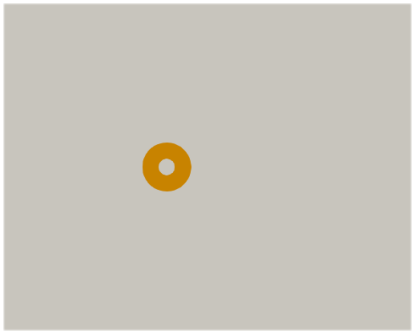


Wireframe visualization (**cylinder CM**)

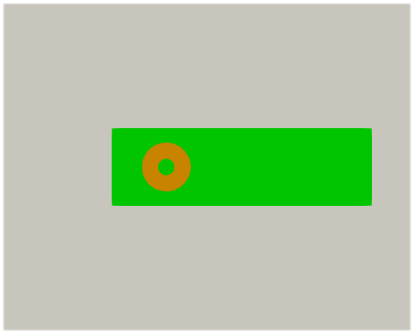


The grammar of overset meshes in OpenFOAM

- **About the order of operations when merging meshes.**
  - In theory, it does not matter the order of the merge operations.
  - At the end, all **CM** should be merged into a single directory.
  - In this case, the **CM cylinder** and **refinementZone** are merged into the **CM** all.
  - The **zoneID** is assigned after merging the meshes.
  - It is highly recommended that the **oversetPatch** be the first one in the boundary file.



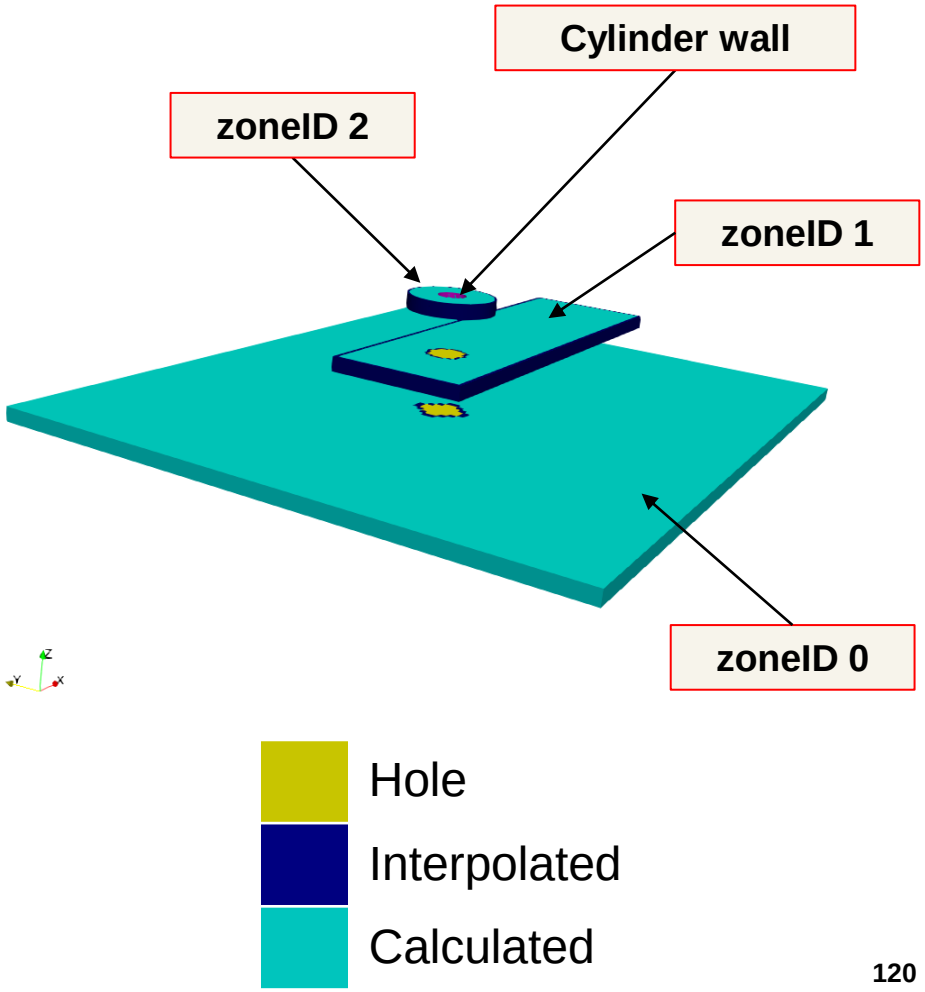
First merge operation → **cylinder** + **all**



Second merge operation → **refinementZone** + **previous merged mesh**

The grammar of overset meshes in OpenFOAM

- **About the zoneID priority (or grid priority).**
  - The **zoneID** defines the order of the hole cutting operations on the component meshes.
  - High **zoneID** values, means high priority. That is, that **CM** will cut or imprint lower priority levels.
  - In this case, the **cylinder** mesh has a **zoneID** equal to 2, the **refinementZone** mesh has a **zoneID** equal to 1, and the **all** mesh (background) has a **zoneID** equal to 0.
  - Therefore, the **cylinder** mesh (the **wall**) will cut meshes **refinementZone** and **all**, the mesh **refinementZone** will cut the mesh **all** (if there are **walls**), and so on.
  - Different grid priorities will give you different overset assemblies and interpolation stencils, this must be carefully planned.
  - Remember, the Chimera holes are computed using walls, so if there are no walls, there are no holes.

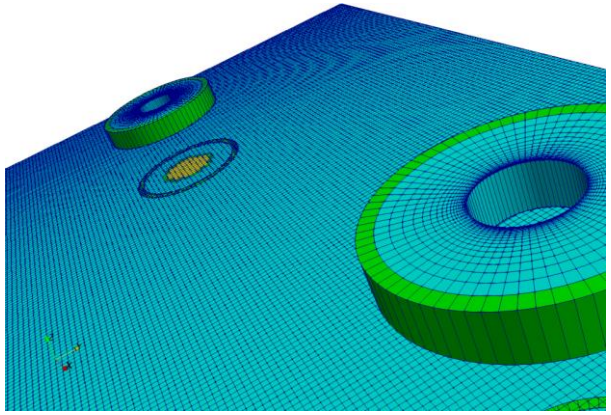


Overset meshes in OpenFOAM

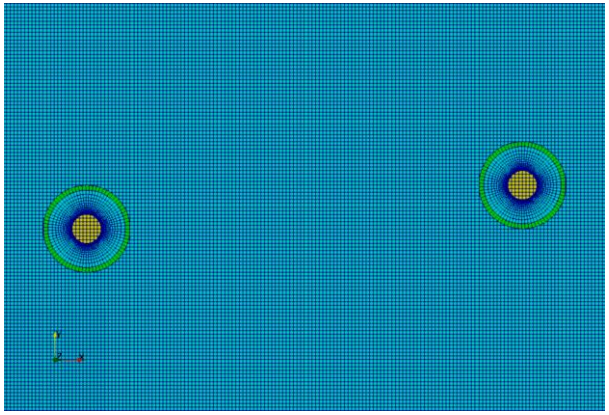
The grammar of overset meshes in OpenFOAM

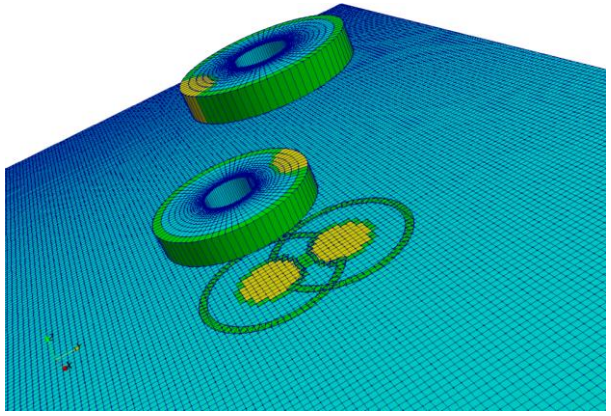
- Multiple bodies undergoing relative motion – Cell types (**cellTypes**) and zones identification (**zoneID**).
- The cell types are recomputed every time-step.

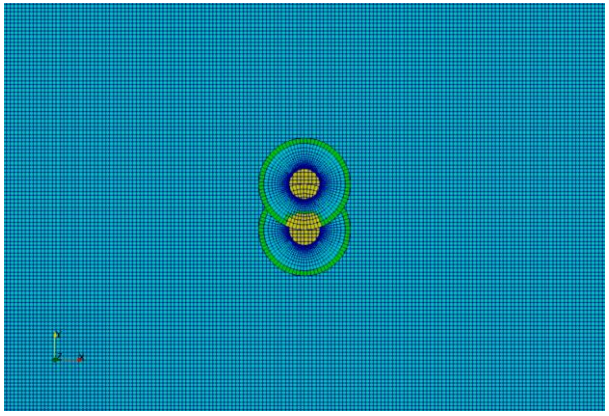
http://www.wolfdynamics.com//wiki/of_conf2019/f4.gif



http://www.wolfdynamics.com//wiki/of_conf2019/f3.gif







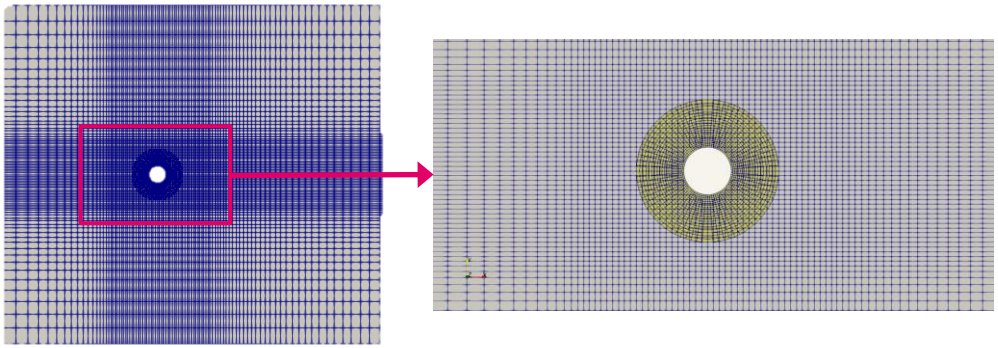
- In this case, the order of the **zoneID** or grid priorities does not make any difference as the cylinders **CM** are identical.
- But if the cylinders **CM** were different, the grid priorities will result in different chimera holes and interpolation stencils.
- The selection of the grid priorities should be planned in advanced. High grid priority means that the **CM** will cut or imprint lower priority grids.

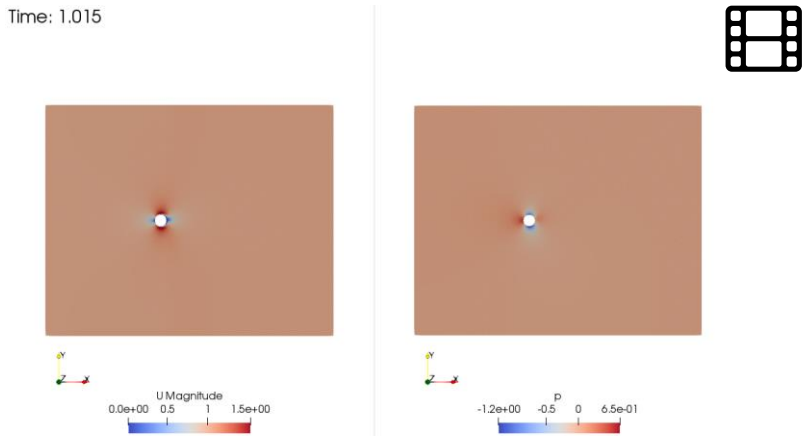
cellTypes	Index
Hole	2
Interpolated	1
Calculated	0

Overset meshes in OpenFOAM

The grammar of overset meshes in OpenFOAM

- Overset meshes simulation workflow in OpenFOAM





http://www.wolfdynamics.com/training/dynamicMeshes/overset5.gif

Step 1 – Generate and merge component meshes
Done by the user

Step 2 – Define overset patches
Done by the user

Step 3 – Assign zones (grid priorities)
Done by the user

Step 4 – Compute stencils, assign cell type, interpolate solution
Done by the solver

Set numerics for overset meshes (overset interpolation type, solution method, CFL number, discretization schemes, corrections, and so on)
Done by the user

Compute and monitor the solution
Done by the solver

Postprocessing (which is more tedious that working with single meshes)
Done by the user

Overset meshes assembly

Compute solution, monitoring, post-processing

Case setup in the overset framework

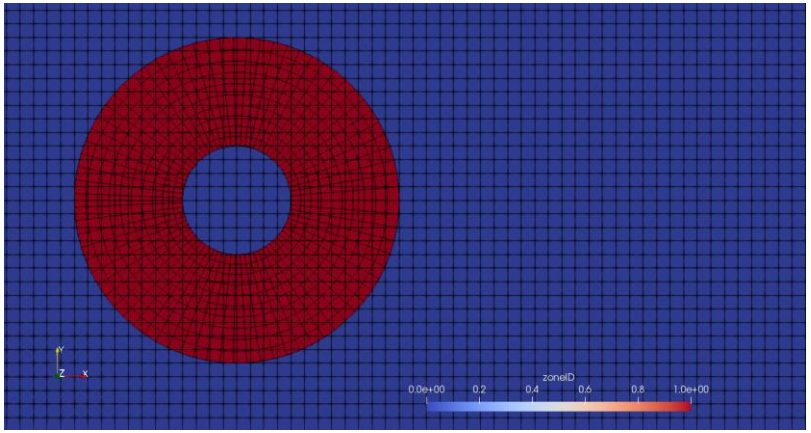
- Fixed 2D cylinder with overset meshes
- Let us run this case to see the typical workflow when using overset mesh. Go to the directory:

```
$PTOFC/overset/cylinderFixed
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Case setup in the overset framework

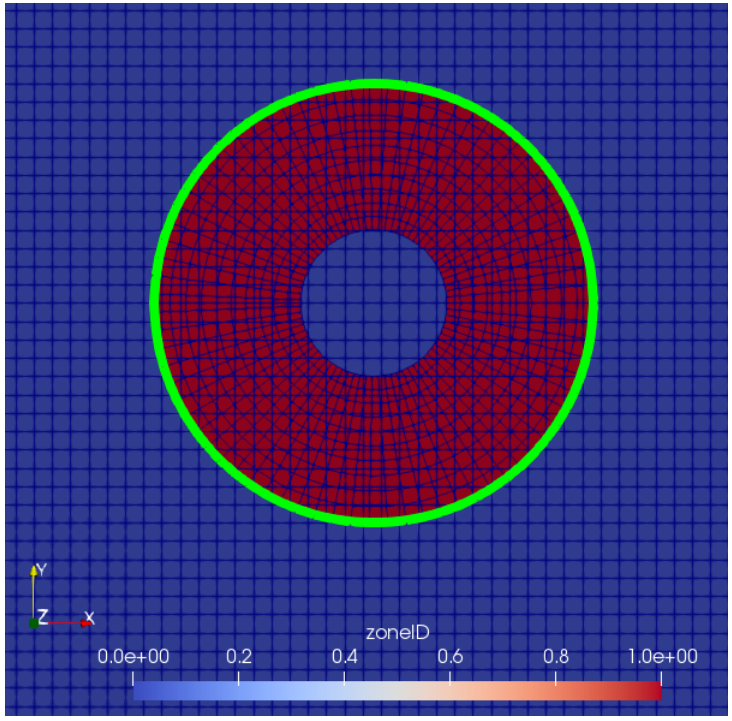
- In the overset framework, a crucial (and mandatory) step is to create zones for handling the different component meshes. This is done using the field **zoneID**.
- To create the zones and assign the field **zoneID** to each zone, we use the `topoSet` and `setFields` utilities:
  - `topoSet`: creates a **cellSet** for each component mesh.
  - `setFields`: assigns to the **cellSet** the proper value of **zoneID**
- The **zoneID** field is also useful for postprocessing overset meshes.
- It is recommended to set the coarsest level to zoneID = 0 (according to developers).
- In overset meshes, when initializing the **zoneID** field, it is recommended to do it independently from other fields (*e.g.*, **alpha.water**), as you may experience problems with the initialization.

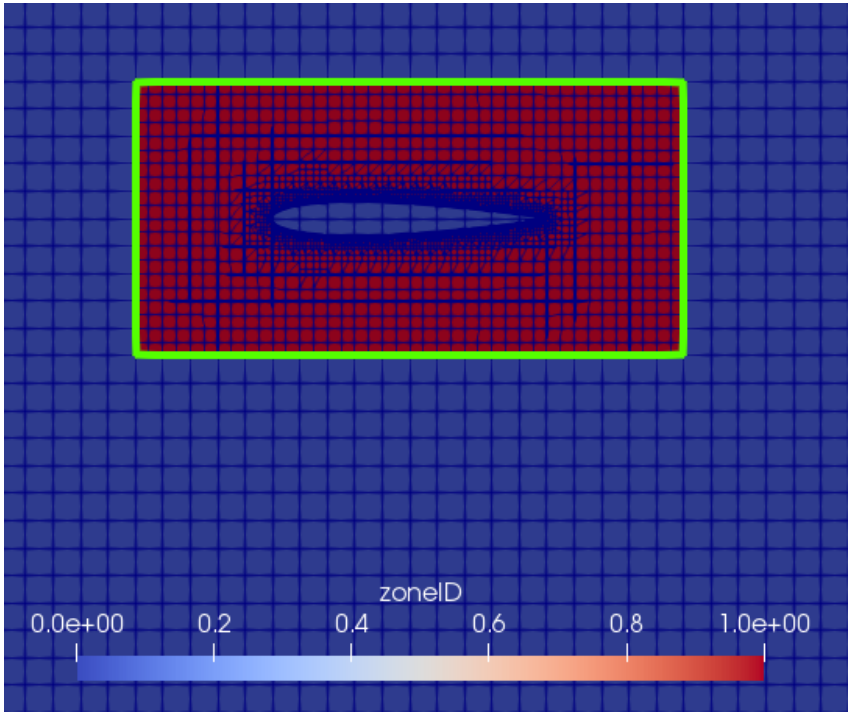


zoneID = 0
zoneID = 1

Case setup in the overset framework

- The new patch type **overset** is used for patches involved in the overset interpolation.
- It can be assigned at meshing time (e.g. in `blockMeshDict`).
- If you import a third-party mesh, it is possible to manually edit the type in the `boundary` file (located in the `constant/polyMesh` directory).
- The overset patches should precede the other patches in the `boundary` file, that is, it should be the first in the list (as recommended by the developers).





Case setup in the overset framework

- You also need to assign the **overset** patch type to the boundary patches in the boundary conditions files located in the ***0*** folder (*i.e.*, `U`, `p`, `k`, `omega`, `epsilon`, `nut`, `alpha.water`, `zoneID`, and so on).

```
boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    yourOversetPatchName
    {
        type        overset;
        value       $internalField;
    }

...
...
...
```

This entry is optional. It is used to automatically assign base type patches(e.g. empty, symmetry)

Specification of overset patch type

Dummy value for paraview.
Be careful to made the distinction between scalar and vector fields

Case setup in the overset framework

- In the *pointDisplacement* dictionary, things are a bit different:

```
boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    yourOversetPatchName
    {
        patchType       overset;
        type            zeroGradient;
    }

...
...
...
```

This entry is optional. It is used to automatically assign base type patches(e.g. empty, symmetry)

patchType keyword instead of type

type to be assigned is zeroGradient

Running parallel cases in the overset framework

- According to the developers, when using overset mesh it is suggested to choose the **hierarchical** or **simple** decomposition methods instead of scotch:

```
numberOfSubdomains  4;

method          hierarchical;

coeffs
{
        n       (2 2 1);
}
```

Decomposition matrix

- This should give a faster and more robust solution.

Overset meshes in OpenFOAM

dynamicMeshDict in the overset framework

- In the overset framework, the *dynamicMeshDict* is always required, even if the meshes are not moving.
- For static meshes (fixed bodies), this dictionary is defined as follows:

```
dynamicFvMesh      dynamicOversetFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solver          displacementLaplacian;

displacementLaplacianCoeffs
{
        diffusivity     uniform 1;
}


dynamicOversetFvMeshCoeffs
{

}
```

Specification of the overset class

Dummy definition of a mesh motion method. This method is doing nothing as it has not been assigned to a patch or body.

Advanced options related to the overset library. This sub-dictionary can also be used with moving bodies. In this case the entry is empty.

Overset meshes in OpenFOAM

dynamicMeshDict in the overset framework

- For cases with moving bodies, generally you can use the same dictionaries entries used with mesh morphing.
- The main difference is that we need to use the **dynamicOversetFvMesh** library instead of **dynamicMotionSolverFvMesh.**
- For example, to assign a prescribed motion:

```
dynamicFvMesh      dynamicOversetFvMesh;

solver         multiSolidBodyMotionSolver;

multiSolidBodyMotionSolverCoeffs
{
        movingZone
        {
                solidBodyMotionFunction linearMotion;
                velocity   (1 0 0);
        }
}
```

Specification of the overset class

Prescribed motion.
Defined in the same was as for morphing meshes

Overset meshes in OpenFOAM

dynamicMeshDict in the overset framework

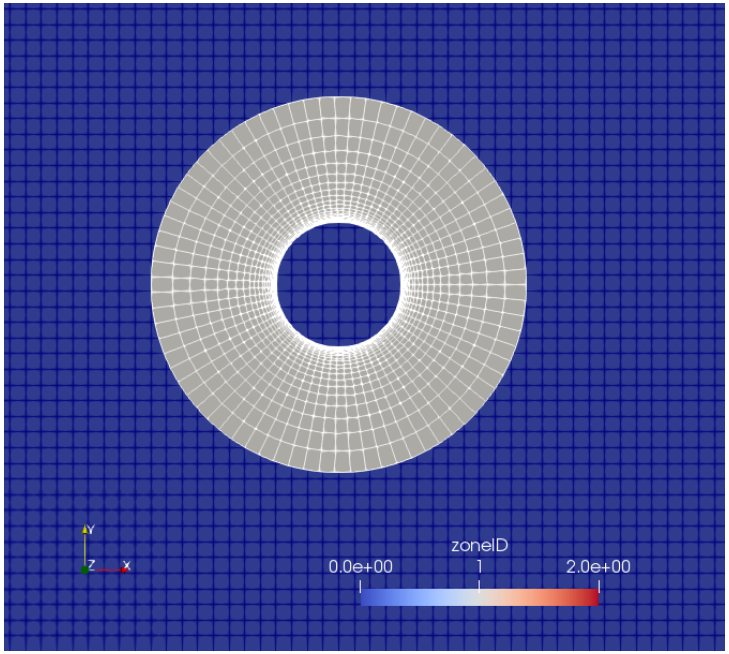
- For cases with moving bodies, generally you can use the same dictionaries entries used with mesh morphing.
- The main difference is that we need to use the **dynamicOversetFvMesh** library instead of **dynamicMotionSolverFvMesh.**
- For example, to use rigid body motion:

```
dynamicFvMesh      dynamicOversetFvMesh;

motionSolverLibs   ("librigidBodyMeshMotion.so");

motionSolver       rigidBodyMotion;

...
...
...
```

Specification of the overset class

Postprocessing in ParaView

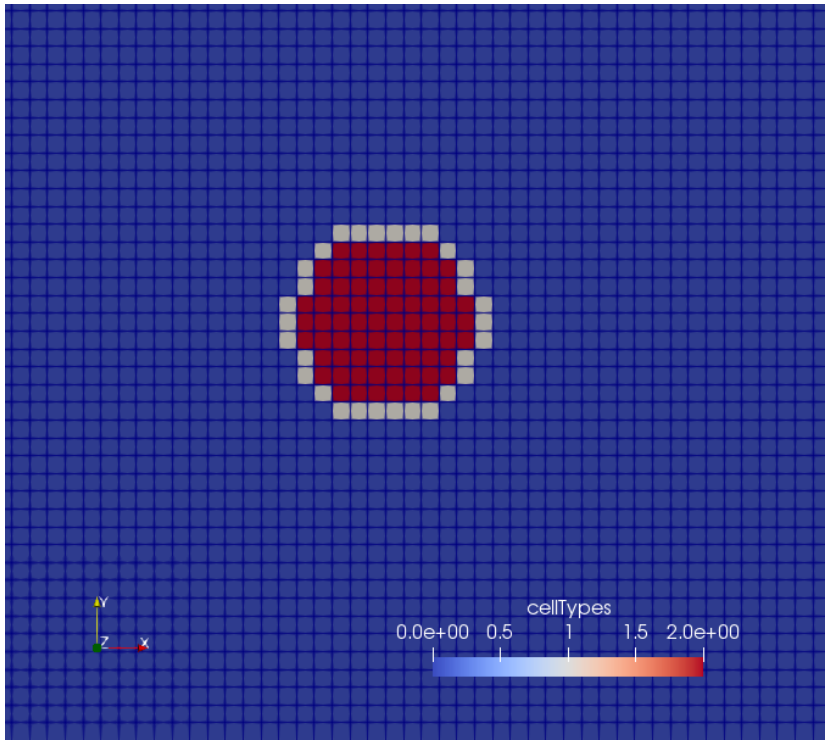
- Visualization of a mesh and solution in ParaView is similar but somehow tricker compared to body fitted mesh cases.
- When launching `paraFoam`, all the meshes will appear as merged.
- In order to visualize separately each component mesh, you can use the **Threshold** filter applied on the variable **zoneID** with the corresponding assigned value,
  e.g., 0 for background, 1 for first inner/moving mesh, 2 for another moving mesh, and so on
- In this way, you can hide/show each mesh layer at your choice.



zoneID = 0
zoneID = 1

Postprocessing in ParaView

- The variable **cellTypes** gives information on the role of the cell in the overset treatment.
- The **cellTypes** values are: 0 → calculated, 1 → interpolated, 2 → masked (not used).

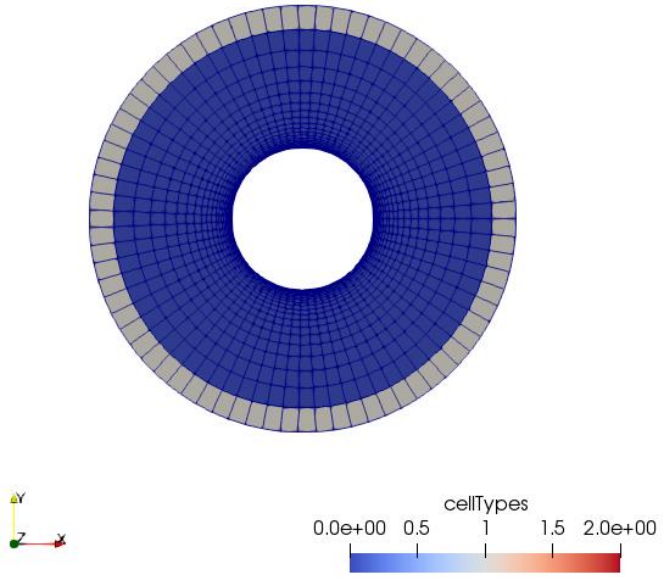


Background mesh

cellTypes = 0 (calculated)
cellTypes = 1 (interpolated)
cellTypes = 2 (masked)

Postprocessing in ParaView

- The variable **cellTypes** gives information on the role of the cell in the overset treatment.
- The **cellTypes** values are: 0 → calculated, 1 → interpolated, 2 → masked (not used).



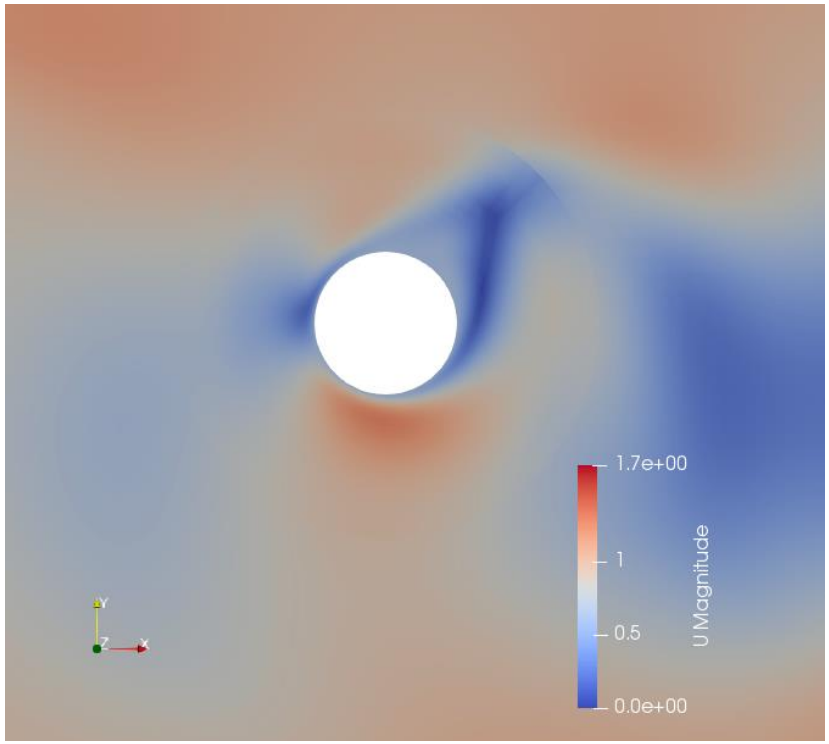
Inner mesh

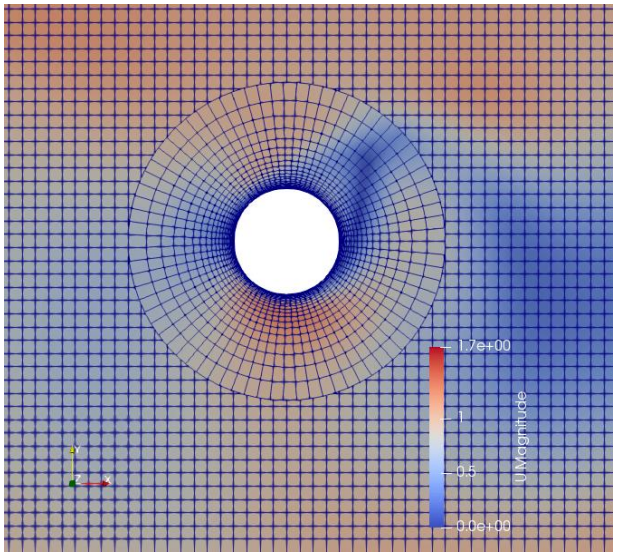
cellTypes = 0 (calculated)
cellTypes = 1 (interpolated)
cellTypes = 2 (masked)

Overset meshes in OpenFOAM

Postprocessing in ParaView

- The variable **cellTypes** gives information on the role of the cell in the overset treatment.
- The **cellTypes** values are: 0 → calculated, 1 → interpolated, 2 → masked (not used).
- Thresholding on this quantity let us remove background cells in holes, for the sake of visualization, so that the final result looks like:





Background mesh (without masked cells) + component mesh

Overset meshes in OpenFOAM

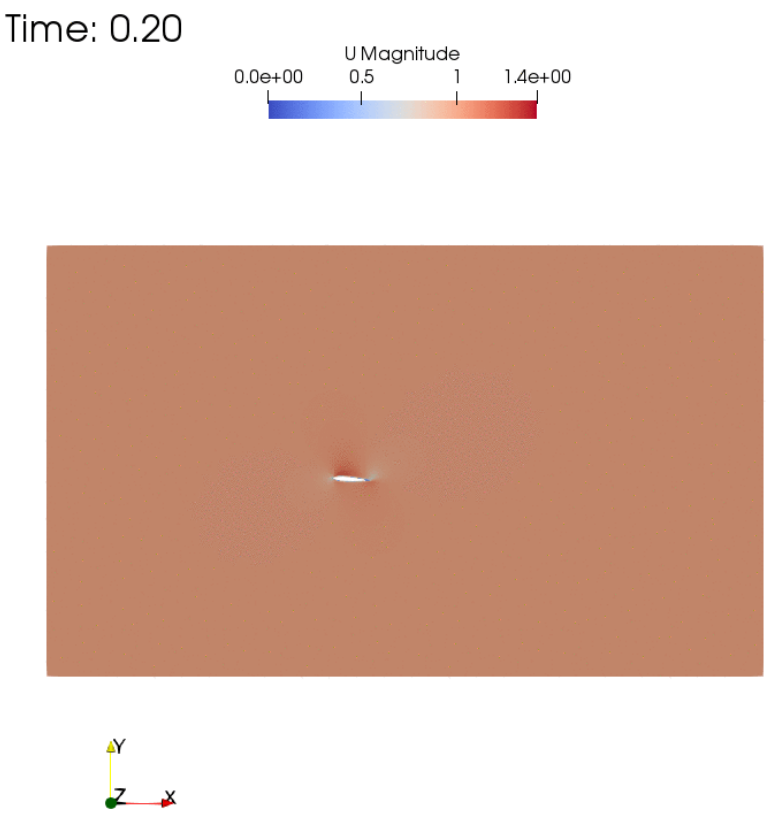
- Flapping airfoil – Prescribed motion with overset meshes
- Let us run this case. Go to the directory:

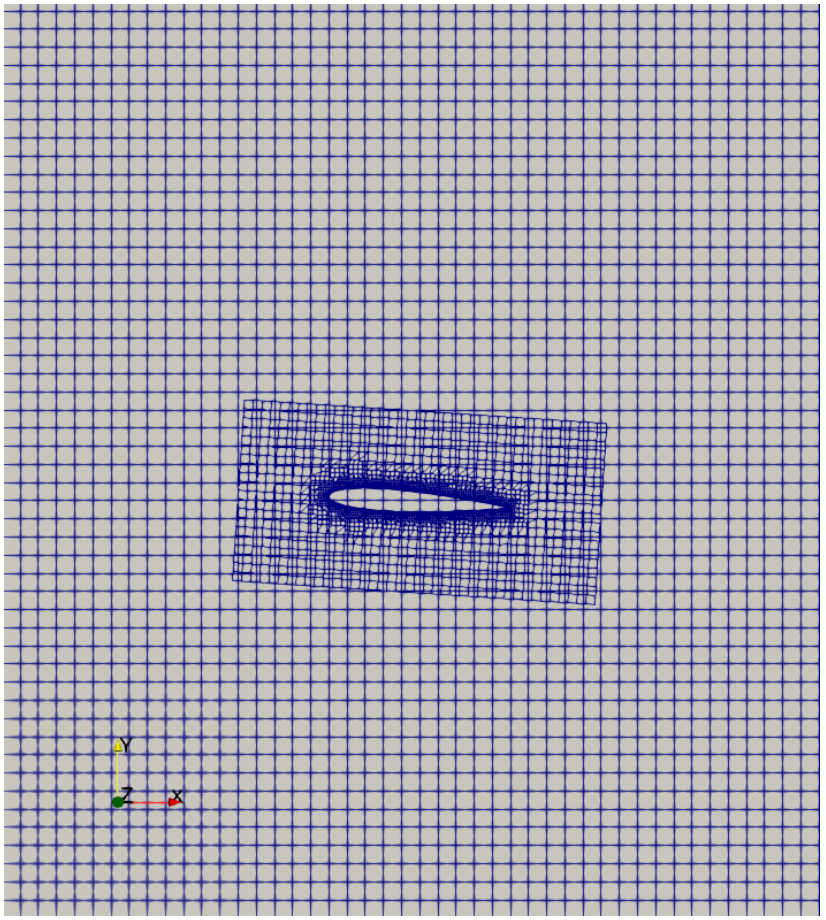
```
$PTOFC/dynamicMeshes/overset_mesh/flapping_airfoil
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Overset meshes in OpenFOAM

Flapping airfoil – Prescribed motion with overset mesh





Flapping airfoil undergoing prescribed heaving and pitching motion.

http://www.wolfdynamics.com/training/dynamicMeshes/overset4.gif

Flapping airfoil – Prescribed motion with overset mesh

- In the same case directory, you will find two versions of this tutorial:
  - **`foil_shmmesh`**: mesh generated with `blockMesh` + `snappyHexMesh`
  - **`foil_fluentmesh`**: mesh generated using an external mesher (ansys mesher and the mesh saved in ansys fluent format)

- Remember, in all cases the overset mesh is created by merging different meshes created separately.

- In this case, we also use the `trasformPoints` utility to shift the inner mesh (this is needed for providing the initial kinematic condition of the airfoil).

Flapping airfoil – Prescribed motion with overset mesh

- In the dictionary *constant/dynamicMeshDict* we specify the motion library, along with the assignment of the overset class to the dynamic mesh treatment.
- Here we use the **multiSolidBodyMotionSolver** which handles multiple moving objects (so that we could add a second flapping airfoil, for instance).
- The related source code can be found here:

dynamicMesh/motionSolvers/displacement/solidBody/solidBodyMotionFunctions

```
dynamicFvMesh      dynamicOversetFvMesh;

dynamicOversetFvMeshCoeffs
{
}

solver          multiSolidBodyMotionSolver;

...
...
...
```

Specification of the overset class

Mesh motion library for prescribed motion of multiple zones

Flapping airfoil – Prescribed motion with overset mesh

- Each moving object has to be associated with a zone (created with `topoSet`), to which we can assign a number of different kinds of motion laws.
- In this case, we employ the **tabulated6DoFMotion** class that can be used for arbitrary motions, providing an input text file in the case directory.
- Details on how to generate this input file containing oscillating motion can be found in the **`gen6DoF`** directory.
- Data appearing in the file *`6DoF.dat`* are (ordering by column):
  time; (translation along x, along y, along z); (rotation around x, around y, around z).

```
...

multiSolidBodyMotionSolverCoeffs
{
    movingZone
    {
        solidBodyMotionFunction tabulated6DoFMotion;
        CofG        (0.33 -0.5 0);
        timeDataFileName "$FOAM_CASE/constant/6DoF.dat";
    }
}
```

Name of zone to put in motion (it was created using the utility topoSet)

Initial position of the center of gravity, with respect to which rotation is performed

Input file for tabulated motion

Flapping airfoil – Prescribed motion with overset mesh

- At this point, we are ready to run the simulation.
- We will use the solver `overPimpleDyMFoam`.
- You will find the instructions of how to run the cases in the file *`README.FIRST`* located in the case directory.
- Before running the simulation, you can check the mesh motion.
- During this check, you can use large time-steps as we are not going to compute the solution, we are only interested in checking the motion.
- To check the mesh motion, type in the terminal:

```
1. $> moveDynamicMesh -noFunctionObjects
```

Overset meshes in OpenFOAM

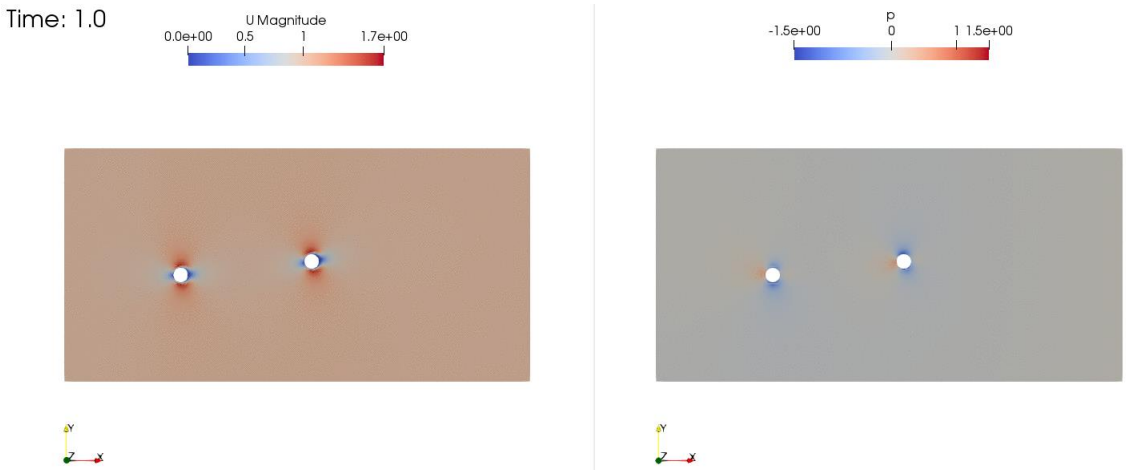
- VIV of two cylinders – Rigid body motion with overset meshes
- Let us run this case. Go to the directory:

```
$PTOFC/dynamicMeshes/overset_mesh/twoCylinders_VIV
```

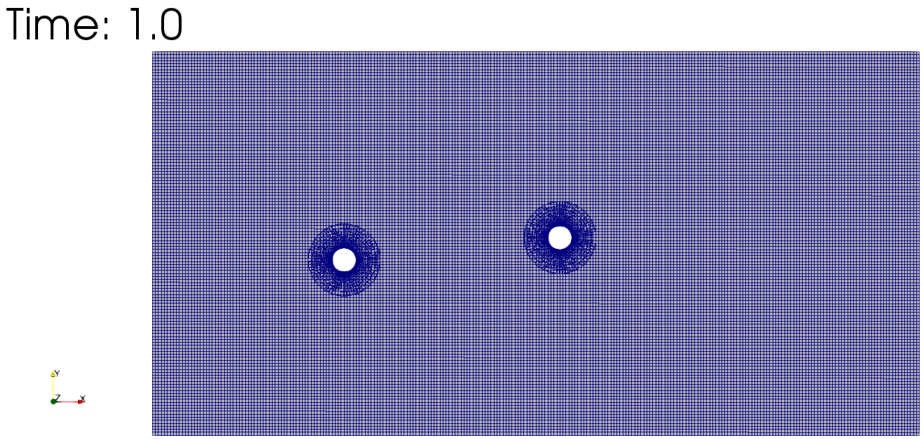
- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Overset meshes in OpenFOAM

VIV of two cylinders – Rigid body motion with overset meshes



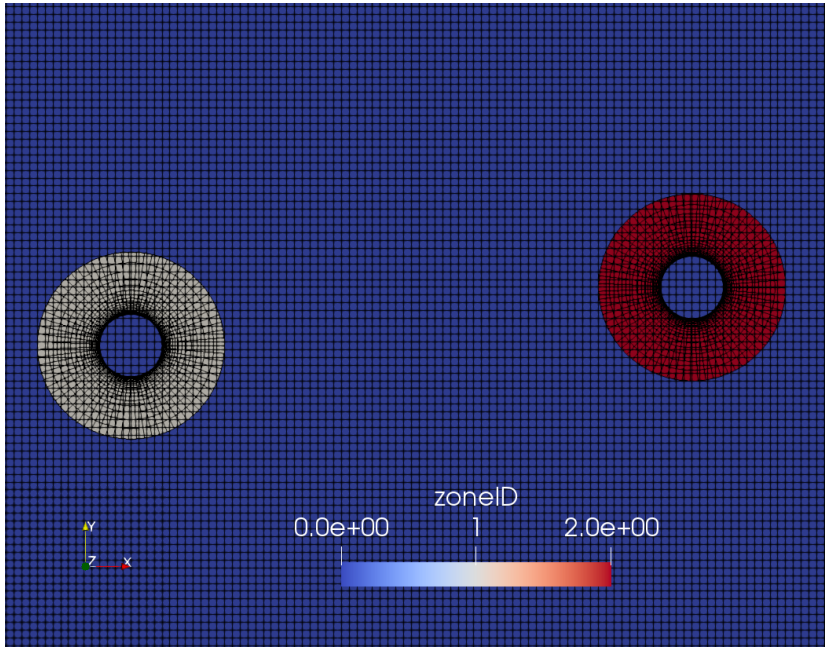
http://www.wolfdynamics.com/training/dynamicMeshes/viv1.gif

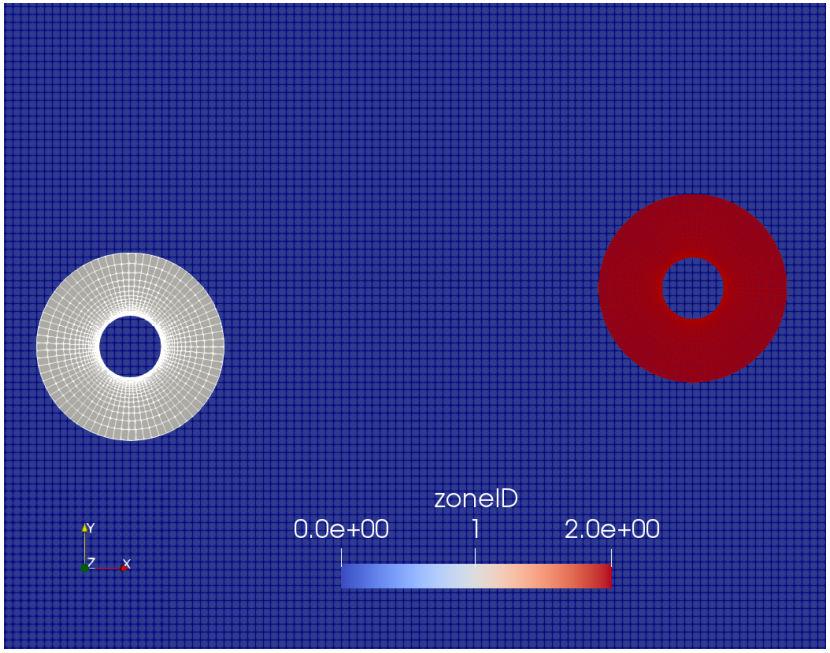


http://www.wolfdynamics.com/training/dynamicMeshes/viv2.gif

VIV of two cylinders – Rigid body motion with overset meshes

- In this tutorial, the meshes will be created using `blockMesh` (for the background mesh) and `extrudeMesh` (for the inner meshes).
- Then we will use the utility `trasformPoints` to translate the second cylinder mesh a given distance with respect to the first cylinder.
- The three meshes will be then merge together using `mergeMeshes` twice.
- Finally, three values of **zoneID** will be assigned using the utility `setFields` (zoneID 0 = background, zoneID 1 = cylinder1, zoneID 2 = cylinder2).





VIV of two cylinders – Rigid body motion with overset meshes

- In this case, both cylinders obey to rigid body dynamics.
- In the dictionary `constant/dynamicMeshDict` we thus need to select a solver library able to handle multiple rigid bodies.
- As in the previous case, we need to assign the **dynamicOversetFvMesh** entry to **dynamicFvMesh**.
- Hence, we choose the **rigidBodyMotion** solver that fits to our goal.

```
dynamicFvMesh      dynamicOversetFvMesh;

motionSolverLibs   ("librigidBodyMeshMotion.so");

motionSolver       rigidBodyMotion;

...
...
...
```

Specification of the overset class

Mesh motion library for rigid body motion (allowing multiple and/or connected bodies)

Overset meshes in OpenFOAM

VIV of two cylinders – Rigid body motion with overset meshes

- Properties of each body are assigned in the corresponding part of the dictionary.

```
bodies
{
        cylinder1          ← First cylinder
        {
                type        rigidBody;
                parent       root;

...
...
...

        }                  ← Second cylinder

        cylinder2
        {
                type        rigidBody;
                parent       root;

...
...
...

      }
```

- Let us see which are the main parameters to set with this kind of rigid body motion solver.

The rigidBodyMeshMotion library

- The **rigidBodyMeshMotion** library handles the dynamics of multiple rigid bodies.
- If you already were familiar the **sixDoFRigidBodyMotion** library, you will find many analogies and few but major differences, the most important ones being the possibility of handling multiple bodies.
- The user can define one or multiple bodies. In case of multiple bodies, kinematic joints between them can be specified as well.
- The rigid motion solver will compute the response of the body (or the system of multiple and connected bodies) to external forces.
- In general, you will use this solver if you are dealing with for mesh-motion of multiple articulated rigid-bodies with joints, restraints and external forces.

- You will find the source code of the **rigidBodyMeshMotion** library in the directory:
  - `OpenFOAM-v2012/src/rigidBodyDynamics`
  - `OpenFOAM-v2012/src/rigidBodyMeshMotion`

- The location is the same for versions 2-12 and 8.

The rigidBodyMeshMotion library

- The dictionary *constant/dynamicMeshDict* contains the selection of the dynamic mesh library and rigid body motion library to be used.
- Here, we also define all the inputs required by the rigid motion solver.

```
dynamicFvMesh       dynamicOversetFvMesh;

motionSolverLibs    ("librigidBodyMeshMotion.so");

solver              rigidBodyMotion;


...
...
...
```

Specification of the overset class

Motion library – Rigid body motion

Solver for mesh motion method

Overset meshes in OpenFOAM

The rigidBodyMeshMotion library

- The dictionary *constant/dynamicMeshDict* (continuation).

```
bodies
{
    cylinder
    {
    type            rigidBody;
    parent          root;
    patches         (cylinder);

    innerDistance   100;
    outerDistance   101;

    centreOfMass    (0.0 0.0 0.0);
    mass            5;
    inertia (1 0 0 1 0 1);

    transform       (1 0 0 0 1 0 0 0 1) (2 2 0);

    report          on;


    solver
    {
        type Newmark;
    }
        ...
```

Other types are: cuboid, sphere, masslessBody, compositeBody

Parent body

Moving patch

Mesh deformation limits when using morphing. Still to be assigned, we put high values so that this will not be active.

Quantity intended in local coordinate system, it is used for transposing the moment of inertia and not to initialize the position of the body

Intended as the inertia tensor with respect to the origin

Tensor of rotation for initial orientation and initial position of the center of rotation

Report on screen position of the body

Rigid body motion solver

You can find the theory behind this library in the following reference:
R. Featherstone. Rigid body dynamics algorithms. Springer, 2008.

Overset meshes in OpenFOAM

The rigidBodyMeshMotion library

- The dictionary *constant/dynamicMeshDict* (continuation).

```
...
...
...

        joint
        {
            type            composite;
            joints
            (
                {
                    type Px;
                }

                {
                    type Rz;
                }
            );
        }
```

Motion DoFs, e.g.:
Px -> translation along x
Py -> translation along y
…
Rz -> rotation around z

- The source code of the joints and restrains is located in the directory
  - **OpenFOAM-2012/src/sixDoFRigidBodyMotion/rigidBodyDynamics**

You can find the theory behind this library in the following reference:
R. Featherstone. Rigid body dynamics algorithms. Springer, 2008.

Overset meshes in OpenFOAM

The rigidBodyMeshMotion library

- The dictionary *constant/dynamicMeshDict* (continuation).

```
...
...
...

    restraints
    {
                                          <-- in local CS
                                          <-- in global CS
        spring1
        {
            type linearSpring;
            refAttachmentPt (0 0 0);
            anchor          (1 1 0);
            stiffness       5.0;
            damping         0.0;
            restLength      1.0;
            body cylinder;
        }

    }
```

Body restraints can be used to apply springs or dampers on the body.

- The source code of the joints and restrains is located in the directory
  - **OpenFOAM-2012/src/sixDoFRigidBodyMotion/rigidBodyDynamics**

You can find the theory behind this library in the following reference:
R. Featherstone. Rigid body dynamics algorithms. Springer, 2008.

The rigidBodyMeshMotion library

- In the dictionary *0/pointDisplacement* we select the body motion.
- For rigid body motion, the body motion is computed by the solver, therefore, we use the boundary condition calculated.

```
cylinder1
{
    type            calculated;
}
```

- As all the walls are moving, we need to use the boundary condition **movingWallVelocity**.
- This is done in the dictionary *0/U*.

```
cylinder1
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

VIV of two cylinders – Rigid body motion with overset meshes

- At this point, we are ready to run the simulation.
- We will use the solver `overPimpleDyMFoam`.
- You will find the instructions of how to run the cases in the file *`README.FIRST`* located in the case directory.
- Remember to adjust the numerics according to your physics.
- Also, have in mind that it is not possible to use the utility `moveDynamicMesh` with rigid body motion, as the motion depends on the forces, which have not been computed yet.

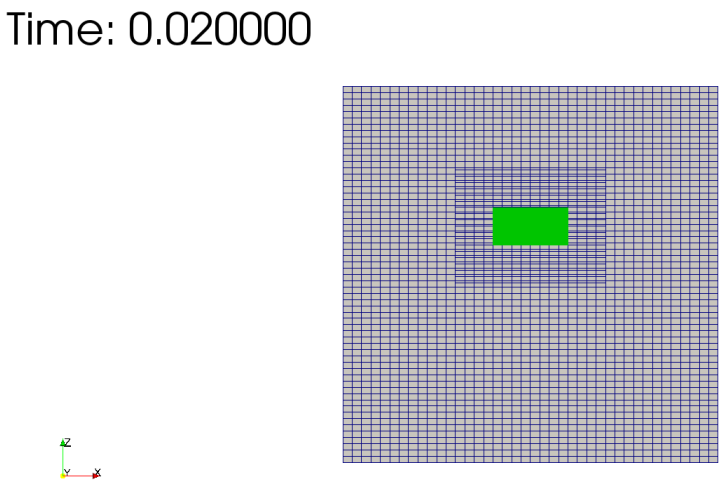
Overset meshes in OpenFOAM

- Floating body – Rigid body motion with overset meshes
- Let us run this case. Go to the directory:

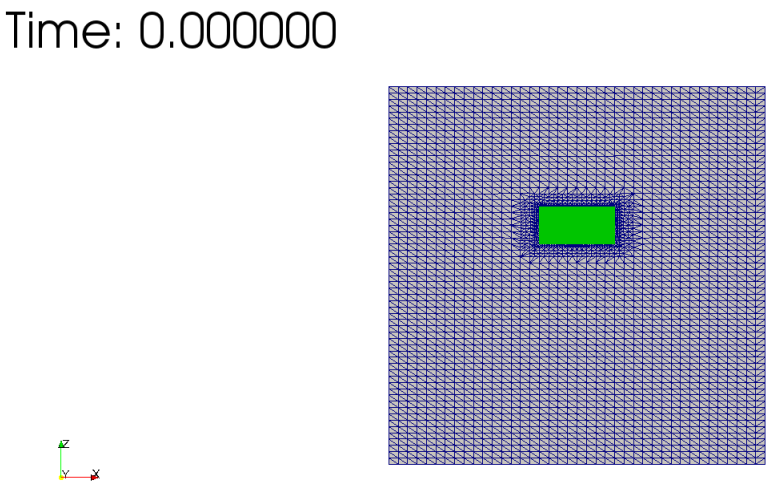
```
$PTOFC/dynamicMeshes/overset_mesh/floatingBody
```

- In the case directory, you will find the `README.FIRST` file. In this file, you will find the general instructions of how to run the case. In this file, you might also find some additional comments.
- You will also find a few additional files (or scripts) with the extension .sh, namely, `run_all.sh`, `run_mesh.sh`, `run_sampling.sh`, `run_solver.sh`, and so on. These files can be used to run the case automatically by typing in the terminal, for example, `sh run_all.sh`. (to run all steps)
- We highly recommend that you open the `README.FIRST` file and type the commands in the terminal, in this way you will get used with the command line interface and OpenFOAM commands.
- If you are already comfortable with OpenFOAM, use the automatic scripts to run the cases.

Floating body – Rigid body motion with overset meshes



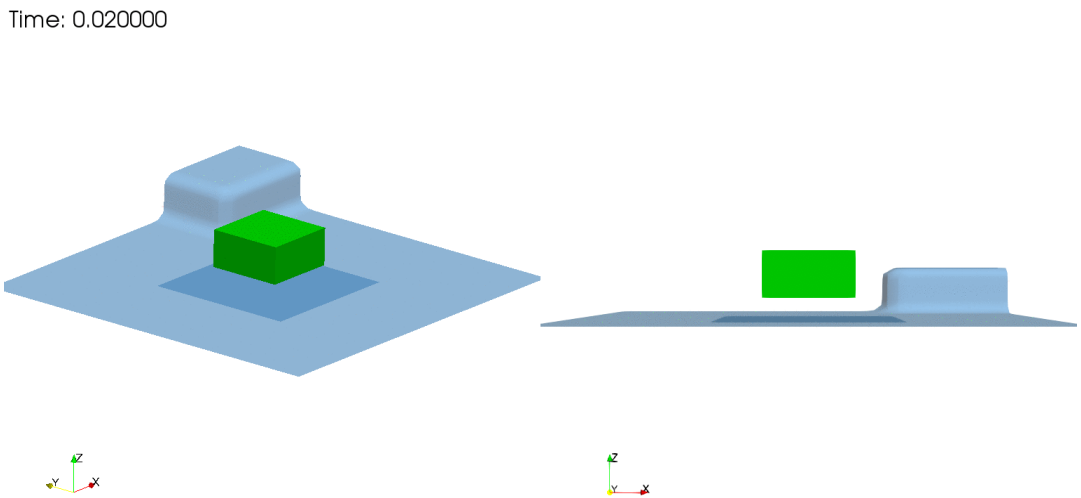
Overset meshes
http://www.wolfdynamics.com/training/dynamicMeshes/overset_rbm1.gif



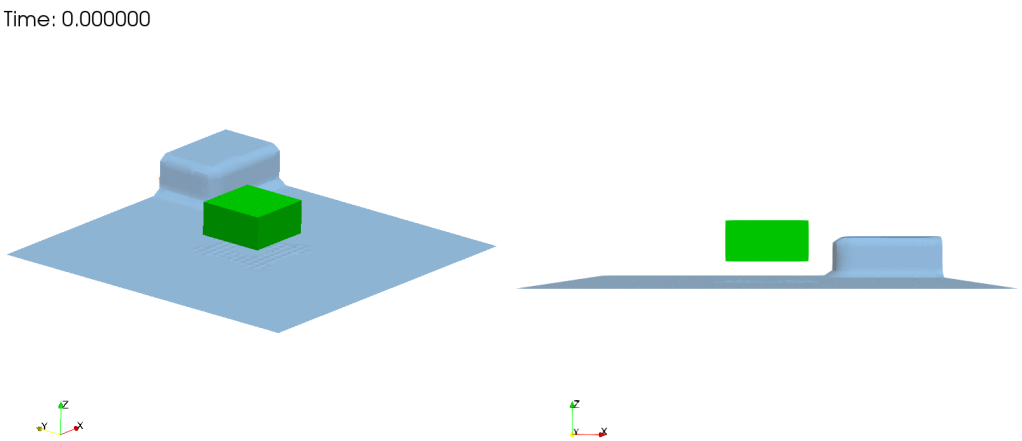
Morphing meshes – Body fitted mesh
http://www.wolfdynamics.com/training/dynamicMeshes/dof1.gif

Overset meshes in OpenFOAM

Floating body – Rigid body motion with overset meshes



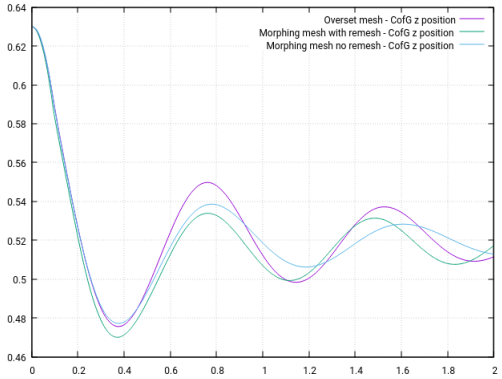
Overset meshes – Water surface visualization
http://www.wolfdynamics.com/training/dynamicMeshes/overset_rbm2.gif



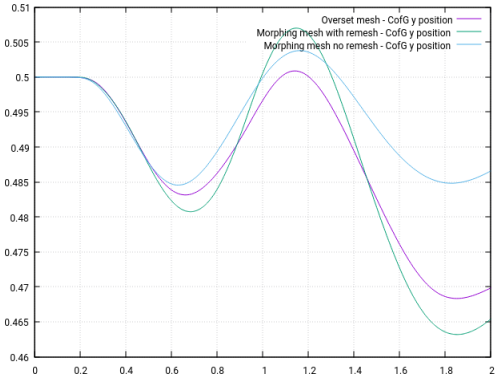
Morphing meshes – Body fitted mesh – Water surface visualization
http://www.wolfdynamics.com/training/dynamicMeshes/dof2.gif

Floating body – Rigid body motion with overset meshes

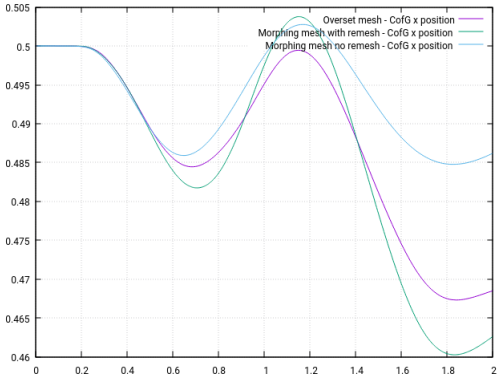
- Comparison of the body dynamics using three different approaches to deal with the rigid body motion.



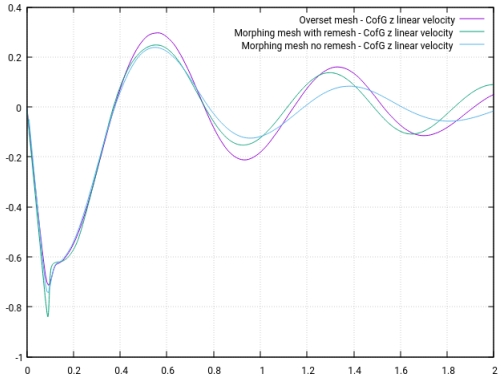
CofG z position vs. Time



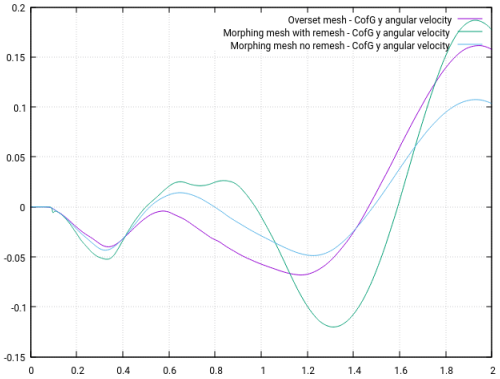
CofG y position vs. Time



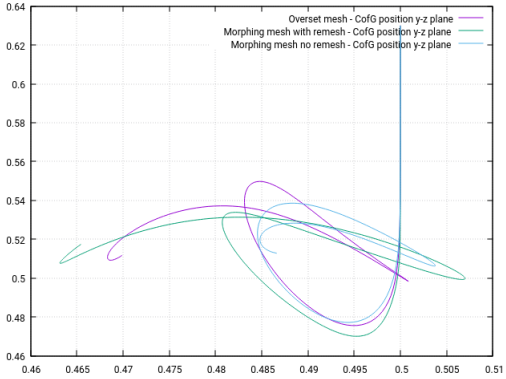
CofG x position vs. Time



CofG z linear velocity vs. Time



CofG angular velocity about axis y vs. Time



CofG position in the plane y-z vs. Time

Floating body – Rigid body motion with overset meshes

- At this point, we can see how the setup of this case in the overset version is rather similar to that using mesh morphing.
- In the dictionary `constant/dynamicMeshDict`, the only difference is in the selection of the dynamic mesh solver.

```
dynamicFvMesh      dynamicOversetFvMesh;

motionSolverLibs   ("libsixDoFRigidBodyMotion.so");

solver             sixDoFRigidBodyMotion;


...
...
...
```

Specification of overset case

Library for rigid body motion

Selection of solver for rigid body motion

Floating body – Rigid body motion with overset meshes

- In the dictionary *0/pointDisplacement* we select the body motion.
- For rigid body motion, the body motion is computed by the solver, therefore, we use the boundary condition calculated.

```
floatingObject
{
    type            calculated;
    value           uniform (0 0 0);
}
```

- As all the walls are moving, we need to use the boundary condition **movingWallVelocity**.
- This is done in the dictionary *0/U*.

```
floatingObject
{
    type            movingWallVelocity;
    value           uniform (0 0 0);
}
```

Floating body – Rigid body motion with overset meshes

- Also, remember that during the mesh creation, we need to assign correctly the overset patches.
- In this case, these are the six faces of the external boundary of the inner mesh, and the specification is provided in the dictionary `blockMeshDict`.

```
…
…
…

boundary
(
        sides
        {
                type overset;
                faces
                (
                        (0 3 2 1)
                        (2 6 5 1)
                        (1 5 4 0)
                        (3 7 6 2)
                        (0 4 7 3)
                        (4 5 6 7)
                );
        }

…
…
…
```

Floating body – Rigid body motion with overset meshes

- And as usual, you will need to adjust the numerics according to your physics.
- In the case directory, you will find the script *extractData*.
- This script can be used to extract the position of the body during the simulation.
- In order to use the *extractData* script, you will need to save the log file of the simulation.
- At this point, we are ready to run the simulation.
- We will use the solver overInterDyMFoam.
- You will find the instructions of how to run the cases in the file *README.FIRST* located in the case directory.

Roadmap

1. ~~Introduction – What are dynamic meshes?~~
2. ~~Adaptive mesh refinement in OpenFOAM~~
3. ~~Sliding meshes in OpenFOAM~~
4. ~~Morphing meshes in OpenFOAM~~
5. ~~Moving meshes in OpenFOAM~~
6. ~~Overset meshes in OpenFOAM~~
7. **Final remarks – General guidelines**

Final remarks – General guidelines

Dynamic/Overset meshes guidelines and tips

- When dealing with prescribed motions, before running the solver test the mesh motion by running the utility `moveDynamicMesh`. If there are **functionObjects** in the *`controlDict`* dictionary, remember to use the option `-noFunctionObjects`, so you do not execute them.
- If you are dealing with rigid body motion, it is not possible to use the utility `moveDynamicMesh` as the motion depends on the forces, which have not been computed yet.
- When walls are moving, remember to always use the **movingWallVelocity** boundary condition.
- To check the mesh courant number, add the entry **checkMeshCourantNo yes**. You can add this option to the **PIMPLE** sub-dictionary in the *`fvSolution`* dictionary.
- For moving bodies, you can use the option **moveMeshOuterCorrectors yes** to gain more stability. This will update the mesh every single outer iteration of the **PIMPLE** loop (with iterative marching enabled). You add this option in the **PIMPLE** sub-dictionary of the *`fvSolution`* dictionary.
- The motion library **solidBodyMotionFunction** specifies the choice of prescribed motion, the source code is located in the directory: **`OpenFOAM-8/src/dynamicMesh/motionSolvers/displacement/solidBody`**
- The following options of prescribed motions are available:
  - axisRotationMotion
  - linearMotion
  - multiMotion
  - oscillatingLinearMotion
  - oscillatingRotatingMotion
  - rotatingMotion
  - SDA
  - tabulated6DoFMotion

Final remarks – General guidelines

Dynamic/Overset meshes guidelines and tips

- Remember, you can specify a variable as an input table. So, for example, if you want to specify omega or velocity as a function of time, you can proceed as follows,

```
omega table
(
        (0      0)
        (0.5    1)
        (1.0    5)
)
```

```
velocity table
(
        (0      (0 0 0) )
        (0.5    (1 0 0) )
        (1.0    (5 0 0) )
)
```

- Moving bodies simulations are intrinsically unsteady, however, it is possible to reach a steady solution if you are interested in finding a trim (equilibrium) condition. In these cases, use the LTS method (local time stepping) for time discretization.
- When dealing with dynamic meshes and using any of the dynamic mesh methods studied, a robust and accurate numerical setup is required.
- Also, as the bodies usually experienced strong accelerations, it is recommended to keep the CFL number below one in order to avoid spurious oscillations.
- For time discretization, the Euler method is preferred over the backwards and CrankNicolson schemes as they may give spurious oscillations with moving meshes.
- If you are not interested in capturing the initial transient, it is recommended to start moving bodies simulations from a previously converged steady simulation (fixed body).

Final remarks – General guidelines

Dynamic/Overset meshes guidelines and tips

- Overset solvers are only available in the OpenFOAM version supported by ESI-OpenCFD. The following solvers are available:
  - `overPotentialFoam`, `overLaplacianDyMFoam`, `overSimpleFoam`, `overPimpleDyMFoam`, `overRhoSimpleFoam`, `overRhoPimpleDyMFoam`, `overInterDyMFoam`
- In order to use the overset solvers, you will need to add the library **liboverset.so** to the *`controlDict`* dictionary.
- In overset meshes, when initializing the **zoneID** field, it is recommended to do it independently from other fields. That is, use two different *`setFields`* dictionaries.
- Also, the order of the **zoneID** is important. It is recommended to assign the **zoneID** 0 to the background mesh (usually the mesh that it is not moving or the mesh holding the external boundary conditions).
- When working with overset meshes, the **GAMG** solver is not supported for pressure. Use **PCG** or **PBiCGStab** instead.
- For turbulence modeling in overset meshes, the **meshWave** method is not supported for wall distance calculation. **Poisson** and **advectionDiffusion** are supported.
- If you are running steady simulation with overset meshes, use tighter under-relaxation factors (for SIMPLE and SIMPLEC).

Final remarks – General guidelines

Dynamic/Overset meshes guidelines and tips

- The overset interpolation method is set in the `fvSchemes` dictionary, the following methods are available (they all are non-conservative):
  - **cellVolumeWeight** – First order accurate. Fast. It has problems with hole detection.
  - **inverDistance** – Second order accurate. Bounded. Computationally expensive.
  - **leastSquares** – Second order accurate. Might become unbounded. Computationally expensive. Good accuracy.

- To print detailed information about the interpolation in overset meshes, you can add the following entry to the `controlDict` dictionary:

```
DebugSwitches
{
        overset 1;
}
```

- In overset meshes, it is highly recommended to use the hierarchical decomposition method when running in parallel.
- In overset meshes, cell size close to the overset patch should be of the same size to minimize interpolation errors.

Final remarks – General guidelines

Dynamic/Overset meshes guidelines and tips

- It is recommended to use explicit interpolation for the turbulence variables (k, omega, epsilon, nut and so on), and the volume of fraction field (alpha.*). This is done in the *fvSchemes* dictionary as follows:

```
oversetInterpolationRequired
{
        k;
        omega;
        alpha.water;
}
```

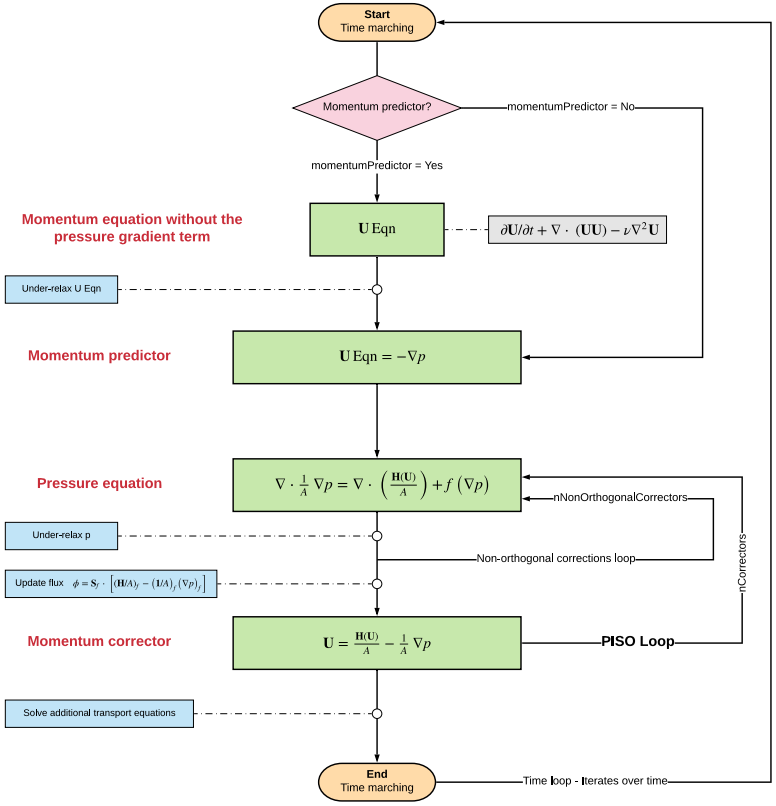
- If the **oversetInterpolationRequired** entry is empty, means full implicit interpolation of all fields.
- When running transient simulations with overset meshes, use the Euler method with a CFL number below 1, as backwards and CrankNicolson schemes may give spurious oscillations.
- In overset meshes, the time-step must be small enough to accommodate for a sequential change of the cell type from blocked to interpolated and then calculated. Therefore, the mesh motion CFL number should be kept ideally below 1.
- In overset meshes, there should be at least 5 or more cells between body patches in order to construct a good interpolation stencil. Cells next to a patch are blocking the flow and cells next to the overset patch are used to interpolate the solution.
- Place the overset interface appropriately, preferably where the field variables do not change much. Avoid strong pressure gradient at the overset patches.

Final remarks – General guidelines

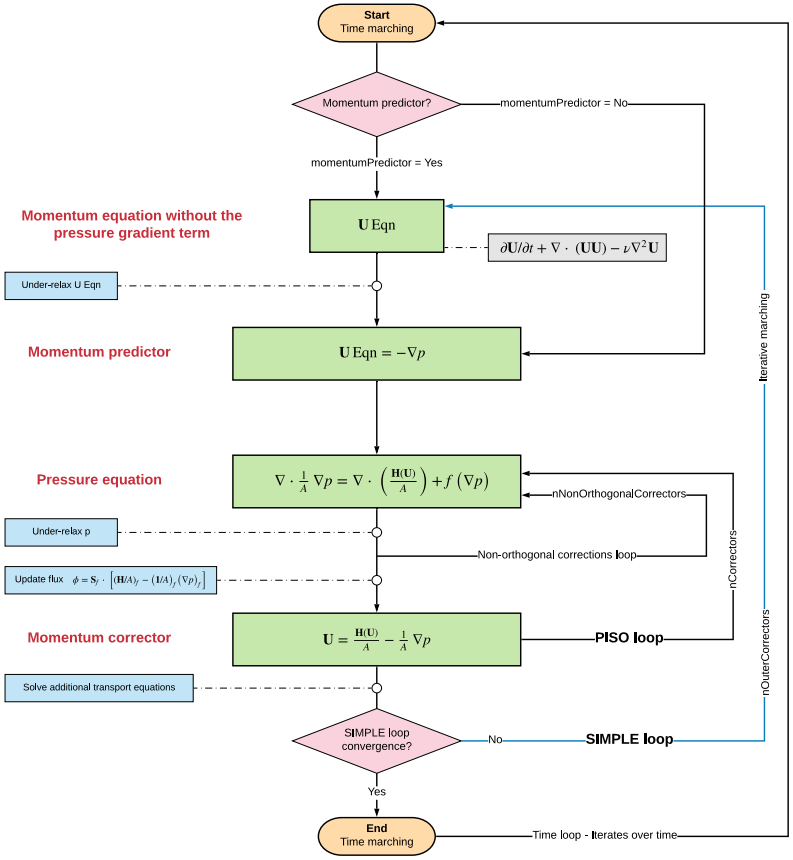
Dynamic/Overset meshes guidelines and tips

- We may sound like a broken record on this, but when dealing with dynamic meshes a robust numerical setup is required. Use iterative marching for the P-V coupling (PIMPLE in OpenFOAM) and do minimum two iterations. For best results, do at least five iterations (in our personal experience).

PISO with non-iterative marching (NITA)



PISO with iterative marching (ITA) – PIMPLE



Thank you for your attention

- We hope you have found this training useful and we hope to see you in one of our advanced training sessions:
    - OpenFOAM® – Multiphase flows
    - OpenFOAM® – Naval applications
    - OpenFOAM® – Turbulence Modeling
    - OpenFOAM® – Compressible flows, heat transfer, and conjugate heat transfer
    - OpenFOAM® – Advanced meshing
    - DAKOTA – Optimization methods and code coupling
    - Python – Programming, data visualization, and exploratory data analysis
    - Python and R – Data science and big data
    - ParaView – Advanced scientific visualization and python scripting
    - And many more available on request
- Besides consulting services, we also offer '**Mentoring Days**' which are days of one-on-one coaching and mentoring on your specific problem.
- For more information, ask your trainer, or visit our website http://www.wolfdynamics.com/

Be collaborative, be innovative,

wolf dynamics

multiphysics simulations,
optimization & data analytics

Let's connect

guerrero@wolfdynamics.com

www.wolfdynamics.com