



38TH
INTERNATIONAL
CONFERENCE
ON COASTAL
ENGINEERING

ROME
2024

8-14
SEPTEMBER



UC

ih
FUNDACION
INSTITUTO DE HIDRÁULICA
AMBIENTAL DE CANTABRIA



IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

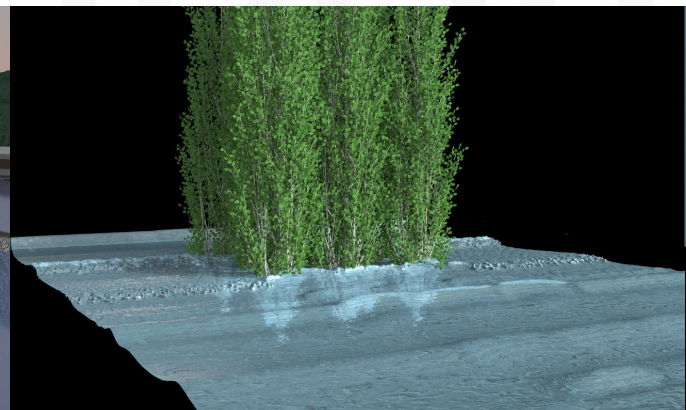
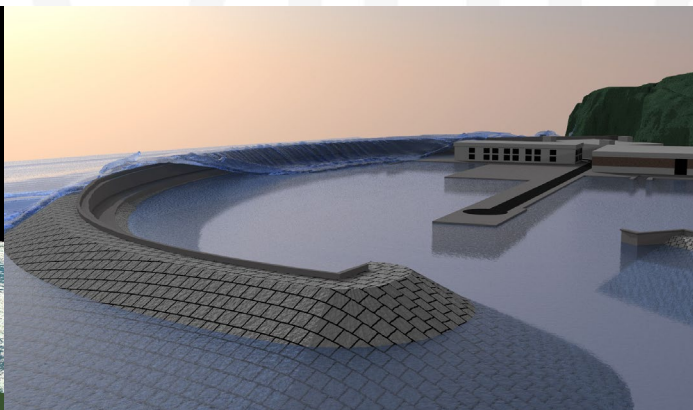
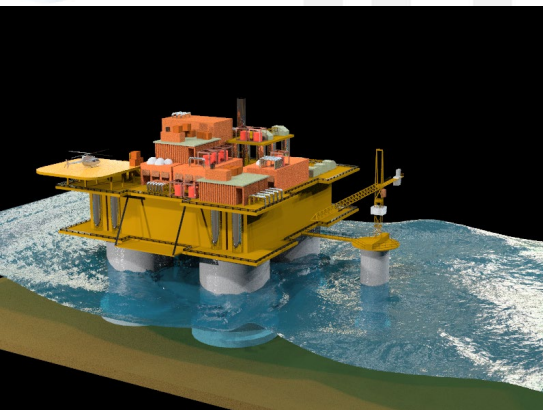
ROMA
TRE
UNIVERSITÀ DEGLI STUDI



IHFOAM applied to Coastal Engineering

Regular waves interaction with a floating structure

Javier L. Lara, Gabriel Barajas, María Maza, Alessandro Romano.



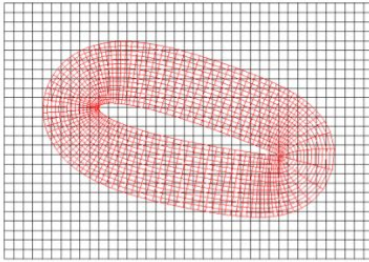
Dynamic mesh handling: OVERSET MESH (v1712 OF)

- 1) Grid generation (overlapping grids: background and floating body mesh)
- 2) Hole cutting
- 3) Interpolation

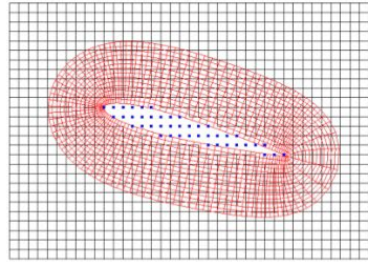
{

Fringe

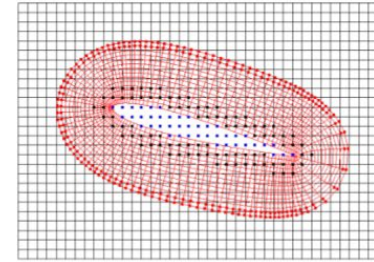
Donor



Background (black) floating (red)



Cutted point (blue)

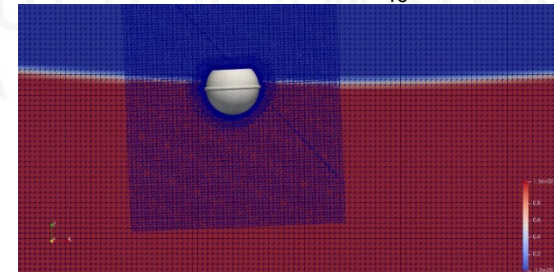
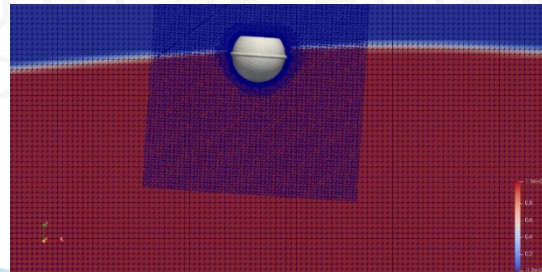


Fringe (black) donor (red)

Copyright:

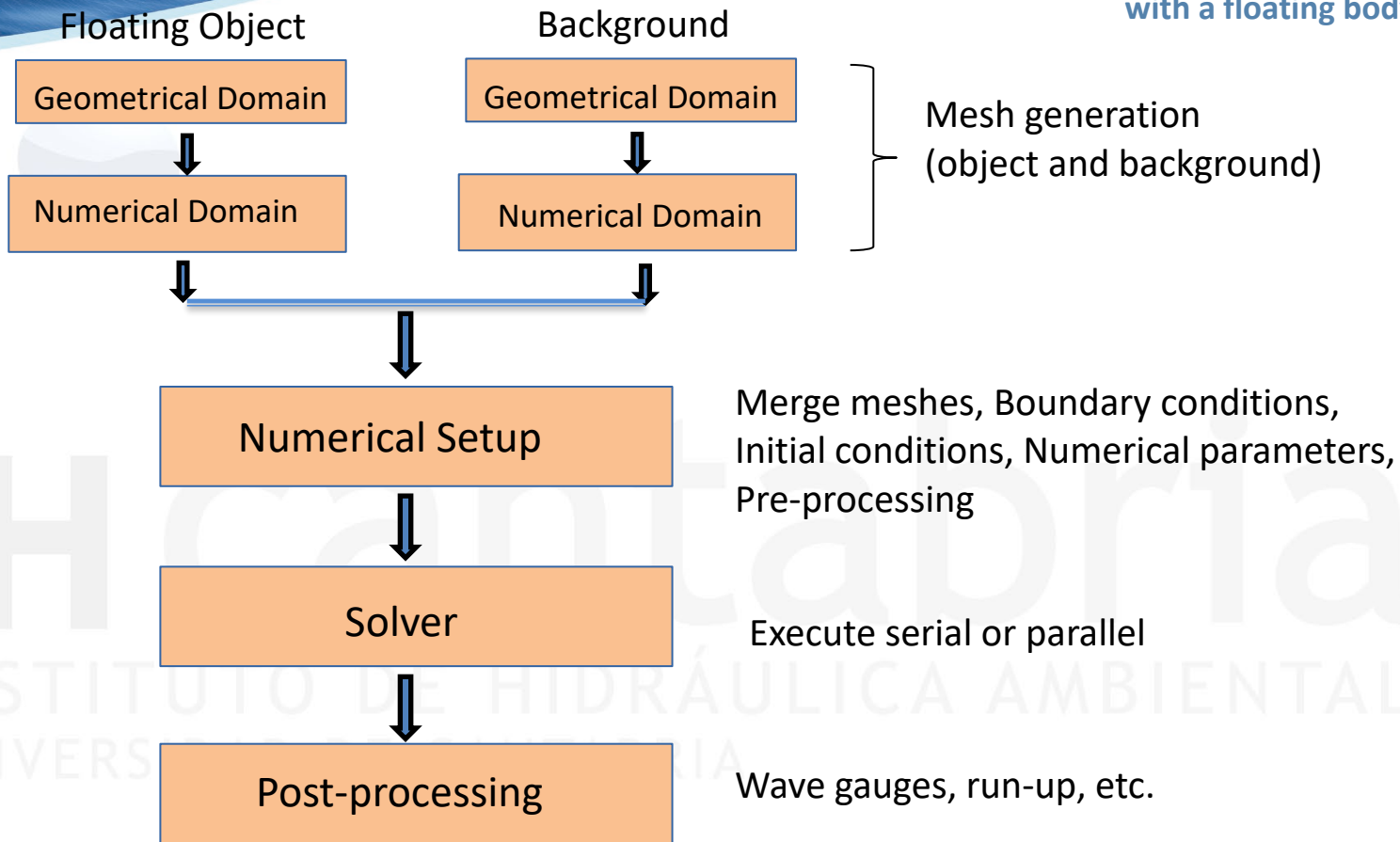
http://celeritassimtech.com/?page_id=13

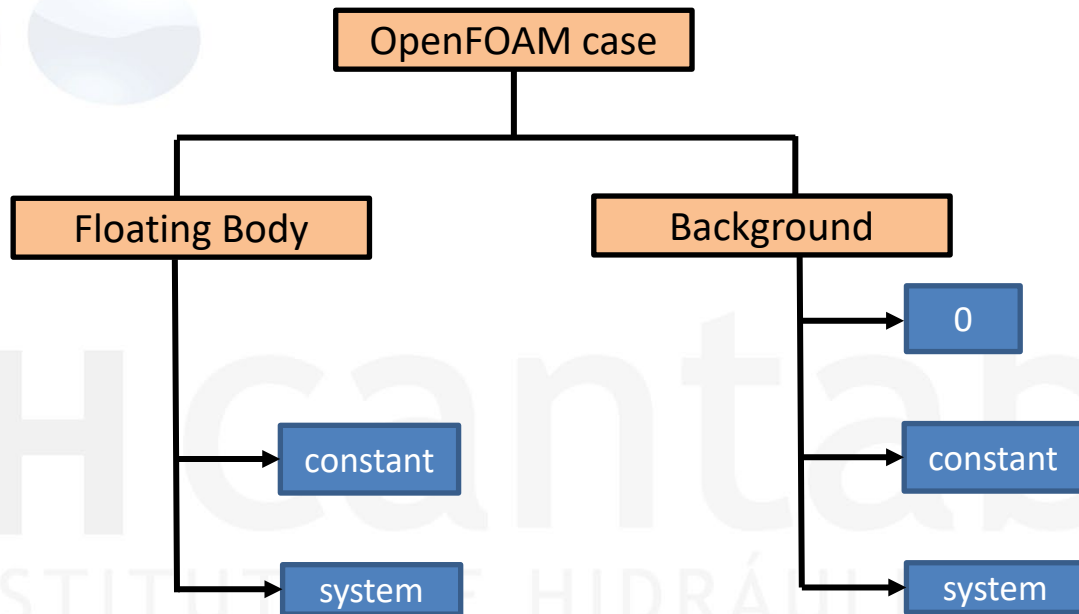
Example



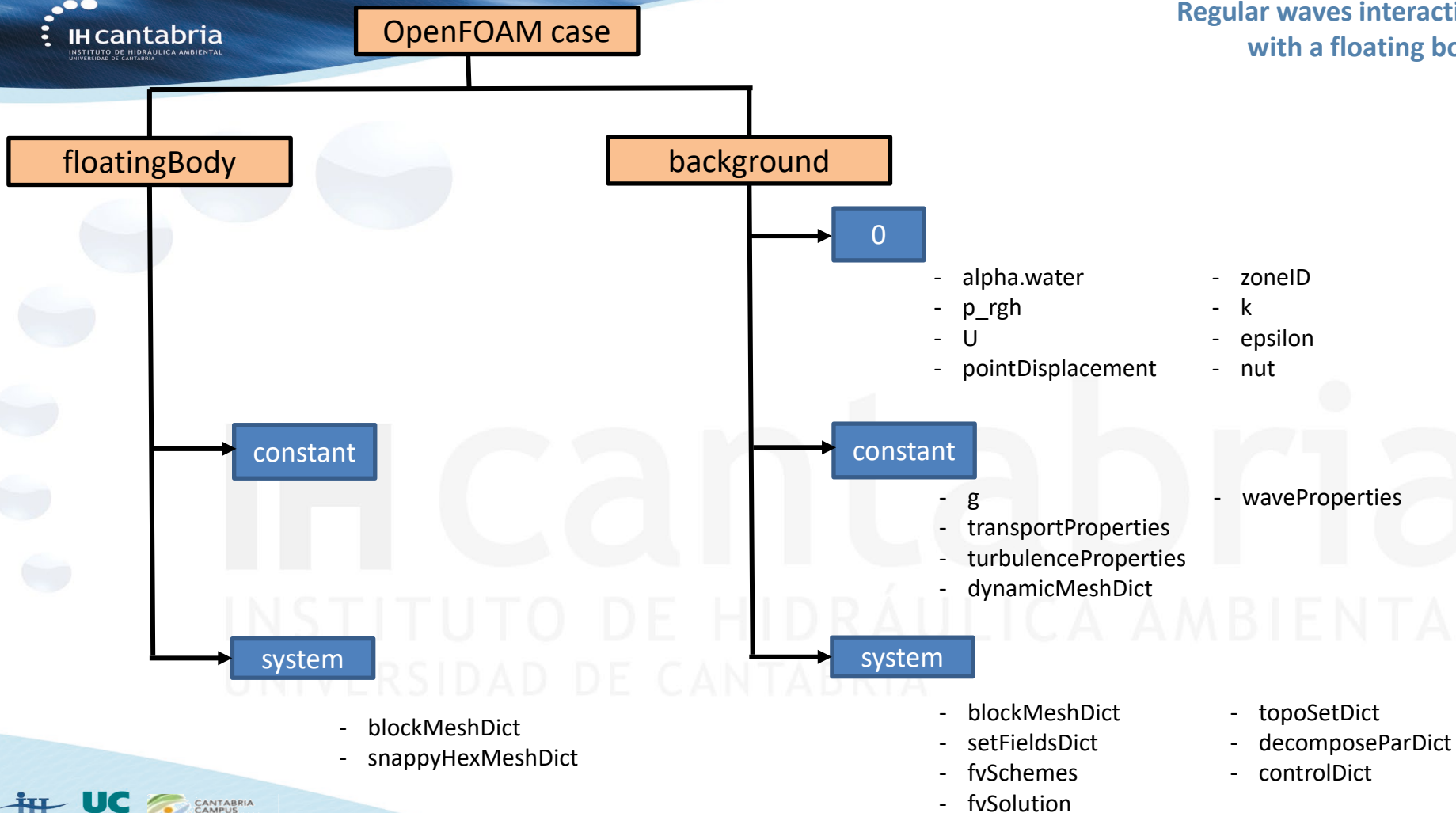
- We will create the case from an existing tutorial:
 - Create a new folder (if it does not exist yet):
`$ mkdir ~/IHFoamCourse/`
 - Copy the 3D case and rename the folder:
`$ cp -r ~/OpenFOAM-v2406/tutorials/multiphase/overInterDyMFoam/floatingBody
~/IHFoamCourse/`
- Set the case:
 - Rename the case:
`$ mv ~/IHFoamCourse/floatingBody ~/IHFoamCourse/overSetWaves`
 - Set the OpenFOAM environment:
`$ source ~/OpenFOAM/OpenFOAM-v2406/etc/bashrc`
 - Ensure that everything you do not need is deleted
`$ cd ~/IHFoamCourse/overSetWaves`
`$./Allclean`

OpenFOAM workflow





IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA



- First, create the mesh for the floating object.
- Edit ***floatingBody/system/blockMeshDict***:

```
scale 1;

vertices
(
  (-0.4 -0.4 -0.4)
  (1.0 -0.4 -0.4)
  (1.0 1.0 -0.4)
  (-0.4 1.0 -0.4)
  (-0.4 -0.4 1.2)
  (1.0 -0.4 1.2)
  (1.0 1.0 1.2)
  (-0.4 1.0 1.2)
);

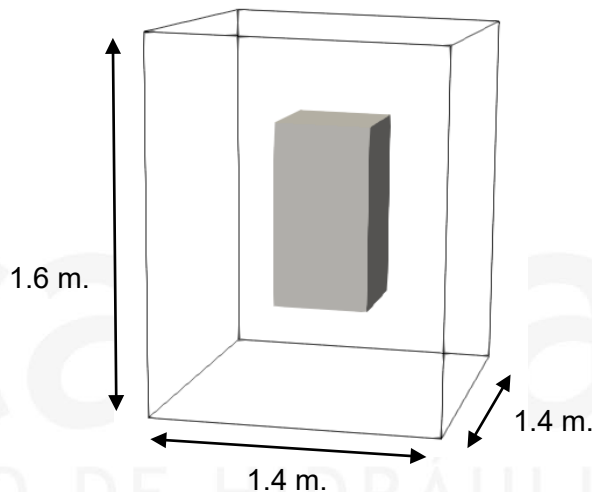
blocks
(
  hex (0 1 2 3 4 5 6 7) (52 52 60) simpleGrading (1 1 1)
);
```

```
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)
    );
  }
);
```

- Create the floating object base mesh:
\$ blockMesh
- Check the floating object base mesh quality:
\$ checkMesh

- Define and create a floating body (using Autocad, Rhino, etc.).
- Check the .stl file; open Paraview, load the .stl file and check that the geometry fits the base mesh:

\$ touch ih.foam && paraview



- Update the case to take into account the new geometry:

\$ mkdir constant/triSurface

\$ cp body.stl constant/triSurface/.

- Use **snappyHexMesh** as mesh generator to take the existing base mesh and remesh it to fit the real geometry of the experiments.
- Copy snappyHexMeshDict from a tutorial:
`$ cp -r ~/OpenFOAM-v2406/tutorials/multiphase/interFoam/RAS/mixerVesselAMI/system/
snappyHexMeshDict ~/IHFoamCourse/overSetWaves/system/.`
- This intermediate mesh is created from the dictionary **system/snappyHexMeshDict**:
 - **CastellatedMesh**:
 - Mesh Refinement in prescribed regions.
 - Detection of the domain (surface and volume).
 - Removal of cells outside the domain.
 - **Snap**:
 - Mesh morphing to follow the provided geometry.
 - Layer addition could also be done.

```
// Which of the steps to run  
castellatedMesh true;  
snap true;  
addLayers false;
```

```
geometry
{
  body.stl
  {
    type triSurfaceMesh;
    name floatingObject;
  }
};
```

→ Definition of the a new boundary (mesh refinement and removal of cells around it).

```
nCellsBetweenLevels 1;
```

→ Minimum number of cells before going to the next level of resolution (Number of buffer layers between different levels.)

```
refinementRegions
{
}
```

→ List of feature edges, that describe Sharp cornes, for refinement.

```
refinementSurfaces
{
  floatingObject
  {
    level (1 2);
  }
}
```

→ Surface based refinements, based on two levels for every surface (the first is the minimum level, the second level is the maximum level).

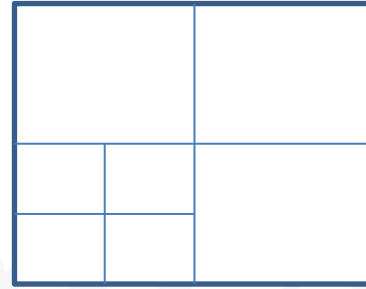
- Refinement levels in OpenFOAM: increase in the refinement level reduces the cell size by half.



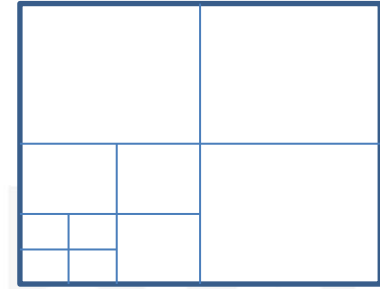
Level 0



Level 1



Level 2



Level 3

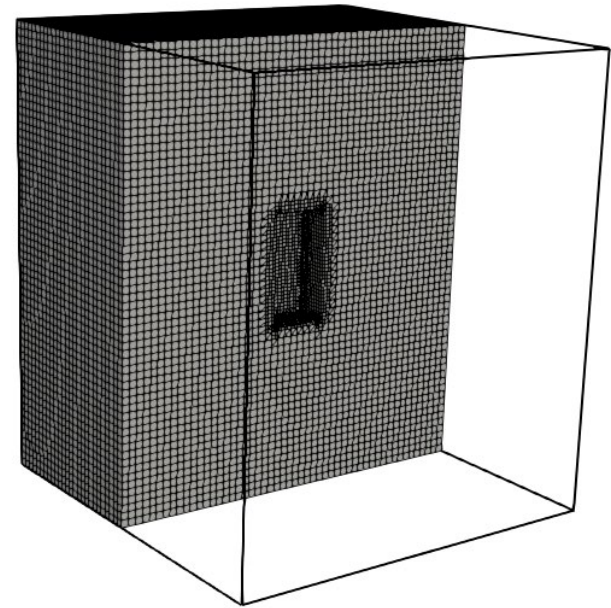
ih cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

```
locationInMesh (0.1 0.1 0.1);
```



Cartesian points (x, y, z) to identify the volumen to retain the final mesh.

- Create the intermediate mesh:
\$ `snappyHexMesh -overwrite`.
- Check the floating object base mesh quality:
\$ `checkMesh`
- Check your final mesh with Paraview:
\$ `paraview`
- Load the ih.foam file. and press “Apply”. (Remember to tick “Skip Zero Time”, as the boundary conditions in the 0 folder have not been updated yet.)



- Next, create the mesh for the background.
- Edit ***background/system/blockMeshDict***:

```
scale 1;

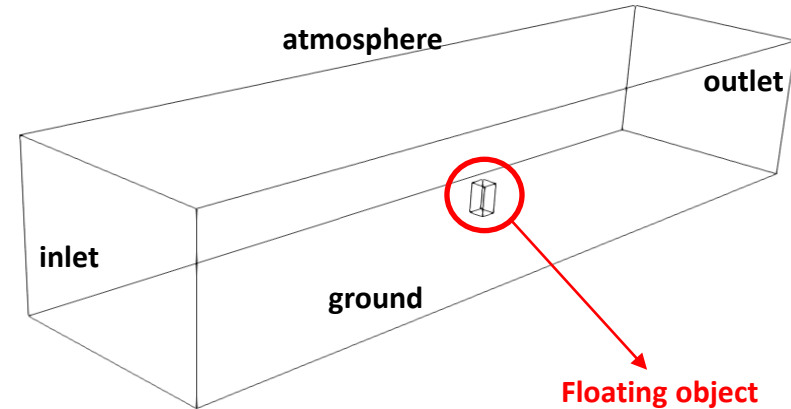
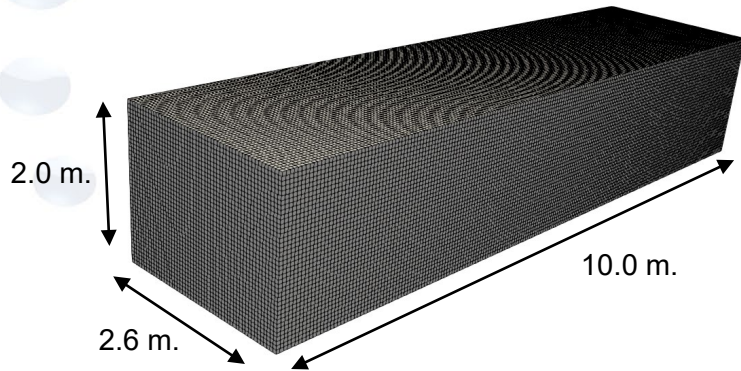
vertices
(
  (-5 -1 -0.5)
  (5 -1 -0.5)
  (5 1.6 -0.5)
  (-5 1.6 -0.5)
  (-5 -1 1.5)
  (5 -1 1.5)
  (5 1.6 1.5)
  (-5 1.6 1.5)
);

blocks
(
  hex (0 1 2 3 4 5 6 7) (200 52 40) simpleGrading (1 1 1)
);
```

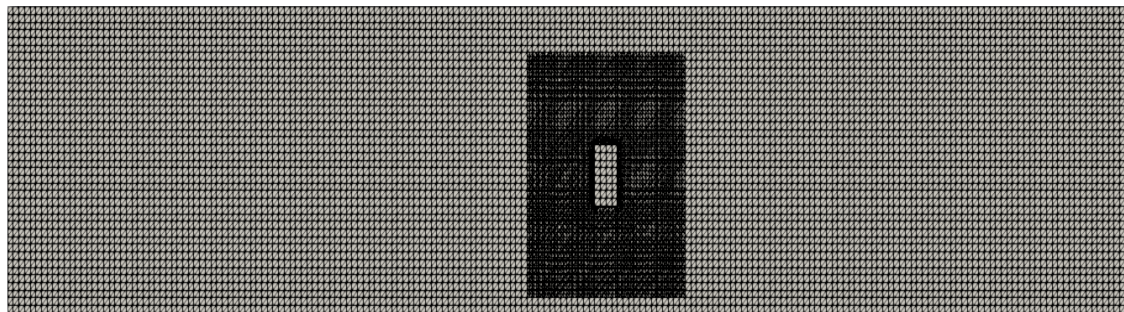
- Create the background base mesh:
`$ blockMesh`
- Check the background base mesh quality:
`$ checkMesh`

```
boundary
(
  oversetPatch
  {
    type overset;
    faces ();
  }
  inlet
  {
    type patch;
    faces
    (
      (0 4 7 3)
    );
  }
  outlet
  {
    type patch;
    faces
    (
      (1 5 6 2)
    );
  }
  ground
  {
    type wall;
    faces
    (
      (0 1 2 3)
    );
  }
  atmosphere
  {
    type patch;
    faces
    (
      (4 5 6 7)
    );
  }
  walls
  {
    type patch;
    faces
    (
      (0 1 5 4)
      (3 2 6 7)
    );
  }
);
```

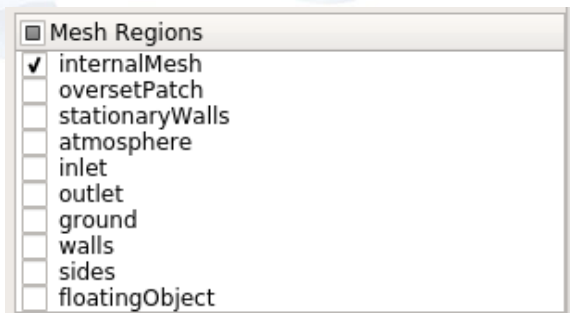

- Check if the floating object and the background mesh are in concordance before merging.
Use Paraview:
\$ touch ih.foam && paraview
- Load **background/ih.foam** file and press “Apply”. (Remember to tick “Skip Zero Time”, as the boundary conditions in the 0 folder have not been updated yet.). Load **floatingBody/ih.foam** too and press “Apply”. Both meshes can be seen together.



- Finally, merge both meshes to create the final and unique mesh:
`$ mergeMeshes ../floatingBody -overwrite`
- Use Paraview to visualize the final mesh:
`$ paraview`
- Load the ih.foam file and press “Apply”. (Remember to tick “Skip Zero Time”, as the boundary conditions in the 0 folder have not been updated yet.)



- The final boundaries can be checked with Paraview (in the *Mesh Regions* dialog box) or they can be checked in the ***constant/polyMesh/boundary*** file.



```

8
(
    overset
    {
        type            overset;
        inGroups         1(overset);
        nFaces           0;
        startFace        1728044;
    }
    inlet
    {
        type            patch;
        nFaces           2080;
        startFace        1728044;
    }
    outlet
    {
        type            patch;
        nFaces           2080;
        startFace        1730124;
    }
    ground
    {
        type            wall;
        inGroups         1(wall);
        nFaces           10400;
        startFace        1732204;
    }
)

```

```

atmosphere
{
    type            patch;
    nFaces          10400;
    startFace       1742604;
}
walls
{
    type            patch;
    nFaces          16000;
    startFace       1753004;
}
sides
{
    type            overset;
    inGroups         1(overset);
    nFaces           17888;
    startFace       1769004;
}
floatingObject
{
    type            wall;
    inGroups         1(wall);
    nFaces           3744;
    startFace       1786892;
}
)

```

- Once the final boundaries are known, update **0.org** folder: VoF(**alpha.water**), velocity (**U**), pressure (**p_rgh**), mesh ID (**ZoneID**) and cell motion (**pointDisplacement**).
- The case is defined as turbulent in:
\$ more constant/turbulenceProperties

```
simulationType RAS;  
  
RAS  
{  
    RASModel      kEpsilon;  
    turbulence     on;  
    printCoeffs    on;  
}
```

- Therefore, the turbulent kinematic energy (**k**), the turbulent dissipation (**epsilon**) and the turbulent viscosity (**nut**) variables must be defined and added to the **0.org** folder.

alpha.water:

```
dimensions      [0 0 0 0 0 0];
internalField    uniform 0;

boundaryField
{
    inlet
    {
        type      waveAlpha;
        value      uniform 0;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      zeroGradient;
    }
    walls
    {
        type      zeroGradient;
    }
    atmosphere
    {
        type      inletOutlet;
        inletValue uniform 0;
        value      uniform 0;
    }
    floatingObject
    {
        type      zeroGradient;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}
```

U:

```
dimensions      [0 1 -1 0 0 0];
internalField    uniform (0 0 0);

boundaryField
{
    inlet
    {
        type      waveVelocity;
        value      uniform (0 0 0);
    }
    outlet
    {
        type      waveVelocity;
        value      uniform (0 0 0);
    }
    walls
    {
        type      slip;
    }
    ground
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    atmosphere
    {
        type      pressureInletOutletVelocity;
        value      uniform (0 0 0);
    }
    floatingObject
    {
        type      movingWallVelocity;
        value      uniform (0 0 0);
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}
```

p_rgh:

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

boundaryField
{
    inlet
    {
        type      fixedFluxPressure;
        value      uniform 0;
    }
    outlet
    {
        type      fixedFluxPressure;
        value      uniform 0;
    }
    ground
    {
        type      fixedFluxPressure;
        value      uniform 0;
    }
    walls
    {
        type      fixedFluxPressure;
        value      uniform 0;
    }
    atmosphere
    {
        type      totalPressure;
        value      uniform 0;
    }
    floatingObject
    {
        type      fixedFluxPressure;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        patchType    overset;
        type          fixedFluxPressure;
    }
}
```

zoneID:

```
dimensions      [0 0 0 0 0 0];
internalField   uniform 0;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      zeroGradient;
    }
    walls
    {
        type      zeroGradient;
    }
    atmosphere
    {
        type      zeroGradient;
    }
    floatingObject
    {
        type      zeroGradient;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}
```

pointDisplacement:

```
dimensions      [0 1 0 0 0 0];
internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    outlet
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    ground
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    walls
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    atmosphere
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    floatingObject
    {
        type      calculated;
        value      uniform (0 0 0);
    }
    overset
    {
        patchType  overset;
        type      zeroGradient;
    }
    sides
    {
        patchType  overset;
        type      zeroGradient;
    }
}
```


k

epsilon

nut

```

dimensions      [0 2 -2 0 0 0];
internalField   uniform 0.00135;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      kqRWallFunction;
        value     uniform 0.00135;
    }
    walls
    {
        type      slip;
    }
    atmosphere
    {
        type      inletOutlet;
        inletValue uniform 0.00135;
        value     uniform 0.00135;
    }
    floatingObject
    {
        type      kqRWallFunction;
        value     uniform 0.00135;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}

```

```

dimensions      [0 2 -3 0 0 0];
internalField   uniform 8.1505e-06;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      epsilonWallFunction;
        Cmu       0.09;
        kappa     0.41;
        E         9.8;
        value     uniform 8.1505e-06;
    }
    walls
    {
        type      slip;
    }
    atmosphere
    {
        type      inletOutlet;
        inletValue uniform 8.1505e-06;
        value     uniform 8.1505e-06;
    }
    floatingObject
    {
        type      epsilonWallFunction;
        Cmu       0.09;
        kappa     0.41;
        E         9.8;
        value     uniform 8.1505e-06;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}

```

```

dimensions      [0 2 -1 0 0 0];
internalField   uniform 0;

boundaryField
{
    inlet
    {
        type      calculated;
        value     uniform 0;
    }
    outlet
    {
        type      calculated;
        value     uniform 0;
    }
    ground
    {
        type      nutkWallFunction;
        Cmu       0.09;
        kappa     0.41;
        E         9.8;
        value     uniform 0;
    }
    walls
    {
        type      slip;
    }
    atmosphere
    {
        type      calculated;
        value     uniform 0;
    }
    floatingObject
    {
        type      nutkWallFunction;
        Cmu       0.09;
        kappa     0.41;
        E         9.8;
        value     uniform 0;
    }
    overset
    {
        type      overset;
    }
    sides
    {
        type      overset;
    }
}

```

- Using **system/topoSet**, generate a set of cells to define the different mesh zones: \$ topoSet

```
actions
(
  {
    name      c0;
    type      cellSet;
    action    new;
    source    regionToCell;
    sourceInfo
    {
      insidePoints ((-0.5 0.3 -0.1));
    }
  }

  {
    name      c1;
    type      cellSet;
    action    new;
    source    cellToCell;
    sourceInfo
    {
      set c0;
    }
  }

  {
    name      c1;
    type      cellSet;
    action    invert;
  }
);
```

→ **regionToCell**: select cells in a region; if started inside, the subCellSet keeps to it, and if outside, stays outside.

→ **CellToCell**: select cells in cellSet.

→ **invert**: select cells not in the cellSet (rest of the domain).

- Update the initial set-up in **system/setFieldsDict**:

```
$ cp -r 0.org 0
```

```
$ setFields
```

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
    volScalarFieldValue zoneID 123
);

regions
(
    boxToCell
    {
        box ( -100 -100 -100 ) ( 100 100 0.5 );
        fieldValues ( volScalarFieldValue alpha.water 1 );
    }
    cellToCell
    {
        set c0;

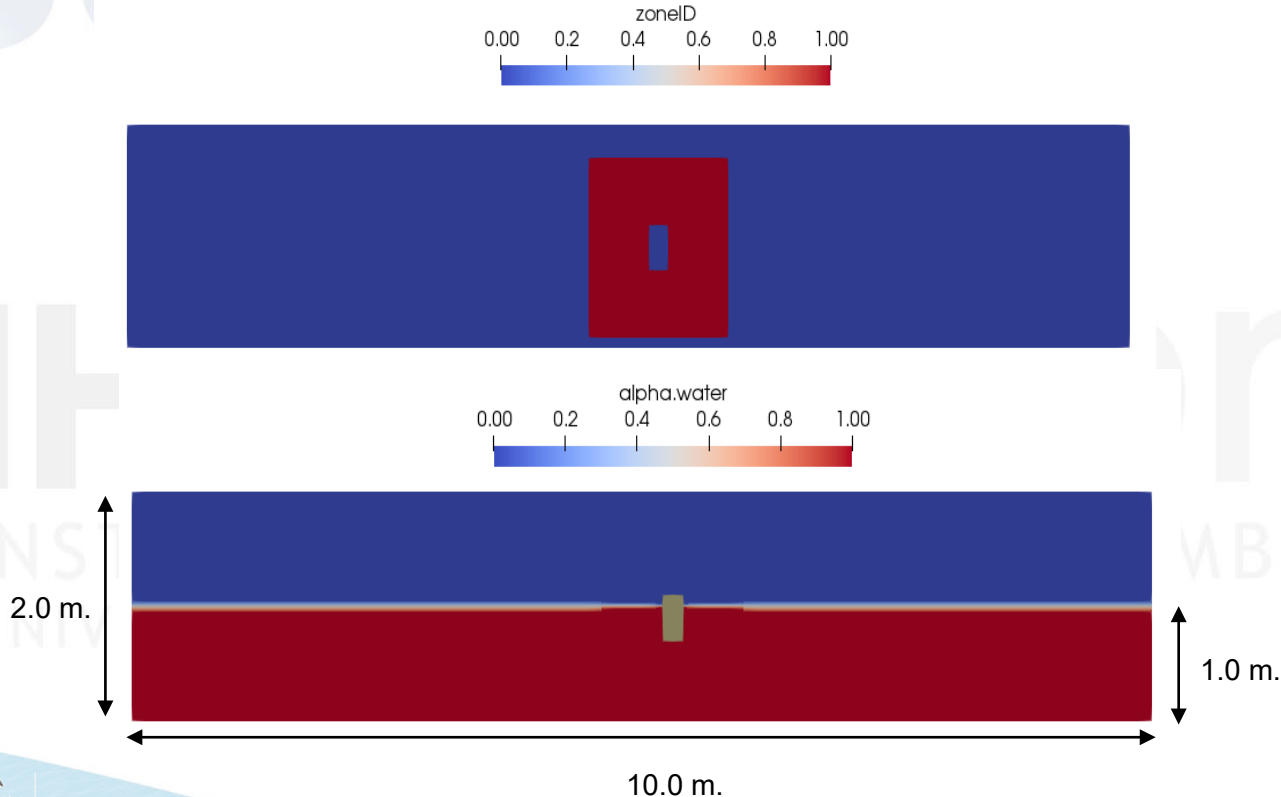
        fieldValues
        (
            volScalarFieldValue zoneID 0
        );
    }
    cellToCell
    {
        set c1;

        fieldValues
        (
            volScalarFieldValue zoneID 1
        );
    }
};
```

Set initial water depth

Set zone identifiers

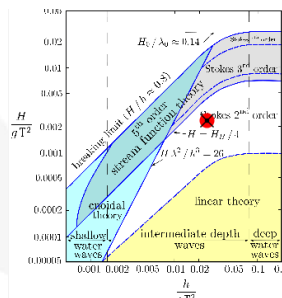
- Open Paraview and plot the initial set-up to ensure everything is correct:
\$ paraview



- Copy the wavePropertiesDict from a tutorial:
`cp ~/OpenFOAM-v2406/tutorials/multiphase/interFoam/laminar/
waveExampleStokesII/constant/waveProperties constant/.`

- Update the wave conditions in **constant/waveProperties**:

- **WaveModel**: Stokes II regular waves
- **nPaddle**: 2 paddles.
- **waveHeight**: $H = 0.1$ m.
- **waveAngle**: 0 (in degrees)
- **rampTime**: 2.0 s (smoothing time).
- **activeAbsorption**: yes (activated absorption at generation).
- **wavePeriod**: $T = 2.0$ s.
- Define 2 paddles to absorb at the outlet.



```
inlet
{
    alpha            alpha.water;

    waveModel        StokesII;

    nPaddle          2;

    waveHeight       0.1;

    waveAngle        0.0;

    rampTime         2.0;

    activeAbsorption yes;

    wavePeriod       2.0;
}
```

```
outlet
{
    alpha            alpha.water;

    waveModel        shallowWaterAbsorption;

    nPaddle          2;
}
```

- Water and air properties are defined in:
\$ more constant/transportProperties

```

phases (water air);

water
{
    transportModel    Newtonian;
    nu                 [0 2 -1 0 0 0 0] 1e-06;
    rho                [1 -3 0 0 0 0 0] 1000;
}

air
{
    transportModel    Newtonian;
    nu                 [0 2 -1 0 0 0 0] 1.48e-05;
    rho                [1 -3 0 0 0 0 0] 1;
}

sigma                [1 0 -2 0 0 0 0] 0.07;
    
```

- Gravity is defined in:
\$ more constant/g

```

dimensions    [0 1 -2 0 0 0 0];
value         ( 0 0 -9.81 );
    
```

- Laminar or turbulent case is defined in:
\$ more constant/turbulenceProperties

```

simulationType    RAS;

RAS
{
    RASModel        kEpsilon;

    turbulence       on;

    printCoeffs      on;
}
    
```


- Define the mesh motion dictionary:
\$ more constant/dynamicMeshDict

```
motionSolverLibs    ("libsixDoFRigidBodyMotion.so");  
dynamicFvMesh       dynamicOversetFvMesh;  
dynamicOversetFvMeshCoeffs  
{  
}  
solver              sixDoFRigidBodyMotion;
```

Import library for the motion solver

Selection of class to handle mesh motion and topology changes

Motion solver

ih cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

```
slxDoFRigidBodyMotionCoeffs
```

```
{
    patches      (floatingObject);
    innerDistance 0.2;
    outerDistance 1.0;

    centreOfMass (0.3 0.3 0.25);

    // Cuboid dimensions
    Lx 0.2;
    Ly 0.2;
    Lz 0.4;

    // Density of the solid
    rhoSolid 700;

    // Cuboid mass
    mass #calc "$rhoSolid*$Lx*$Ly*$Lz";

    // Cuboid moment of inertia about the centre of mass
    momentOfInertia #codeStream
    {
        codeInclude
        #{
            #include "diagTensor.H"
        };

        code
        #{
            scalar sqrLx = sqr($Lx);
            scalar sqrLy = sqr($Ly);
            scalar sqrLz = sqr($Lz);
            os <<
                $mass
                *diagTensor(sqrLy + sqrLz, sqrLx + sqrLz, sqrLx + sqrLy)/12.0;
        };
    };
}
```

Definition of the floating object and the zone within the mesh moves as a rigid body, the zone where the mesh is morphed and the zone with no morphing.

Centre of mass of the floating object

Dimensions of the floating object

Density of the floating object

Mass of the floating object

Moment of inertia of the floating object

```

report      on;
accelerationRelaxation 0.6; }

solver
{
  type Newmark;
}

constraints
{
  fixedLine
  {
    sixDoFRigidBodyMotionConstraint line;
    direction (0 0 1);
  }
}

```

Control the output of the solver (**report**) and definition of a parameter to help maintain the stability of the solver (**accelerationRelaxation** is a direct reduction of the acceleration)

Definition of the body motion **solver** (**Newmark** is a second-order time-integrator)

Definition of a constraint of the motion of the floating object (the **line** constraint defines a direction where the movement of the floating object is only permitted).

IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

Numerical Schemes:

- In **system/fvSchemes**, is the file that sets the numerical scheme for the different terms.

```

ddtSchemes
{
    default      Euler;
}

gradSchemes
{
    default      Gauss linear;
}

divSchemes
{
    div(rhoPhi,U)  Gauss limitedLinearV 1;
    div(U)         Gauss linear;
    div(phi,alpha) Gauss vanLeer;
    div(phiRb,alpha) Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U)))) Gauss linear;

    div(phi,k)     Gauss upwind;
    div(phi,epsilon) Gauss upwind;
}
    
```

Temporal discretization (**Euler** is a first order implicit discretisation scheme).

Gradient derivative terms, that is surface normal gradient terms (**Gauss linear** is second order discretisation scheme)

Divergence terms, such as advection terms and other terms that are often diffusive in nature (**Gauss limitedLinearV 1** is second order, **Gauss vanLeer** is second order and **Gauss upwind** is a first order numerical scheme)

Numerical Schemes:

```
laplacianSchemes
{
  default      Gauss linear corrected;
}

interpolationSchemes
{
  default      linear;
}

snGradSchemes
{
  default      corrected;
}

oversetInterpolation
{
  method       inverseDistance;
}

fluxRequired
{
  default      no;
  p_rgh;
  pcorr;
  alpha.water;
}
```

Laplacian terms, such as the diffusion term in the momentum equation (values between 0 and 1 to handle a non-orthogonal mesh)

Cell to face interpolations of values (linear is second order discretisation scheme)

Component of gradient normal to a cell face (values between 0 and 1 to handle a non-orthogonal mesh)

Define overset interpolation method (***cellVolumeWeight***, ***inverseDistance***, ***leastSquares***, ***trackingInverseDistance***)

Variables needed to calculate fluxes in the pressure equation.

Algorithm control:

- In *system/fvSolutions* are defined the equations solvers, tolerances and algorithms.
- Controls for **MULES**, solver of the VoF equation:
 - **nAlphaCorr**: loops over VoF equation
 - **nAlphaSubcycles**: number of sub-cycles within the VoF equation
 - **cAlpha**: artificial compression velocity.
 - **cAlpha**: artificial compression velocity.
 - **MULESCorr**: switches on semi-implicit MULES.
 - **nLimiterIter**: number of MULES iterations over the limiter.
 - **solver**, **smoother**, **tolerance** and **relTol**: define the solver to solve the matrix equation (symmetric gauss seidel smoother) and tolerances.

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

        MULESCorr       yes;
        nLimiterIter     5;
        alphaApplyPrevCorr no;

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


Algorithm control:

- For each variable solved in the particular equation, the type of ***solver***, ***preconditioner*** and parameters (***tolerance***, ***relTol***, ***maxIter***) that are used by the solver must be defined.
- Normally, the last iteration (variables are solved multiple times within a solution step) is solved with different parameters.

```
"cellDisplacement.*"
{
    solver          PCG;
    preconditioner   DIC;

    tolerance        1e-06;
    relTol           0;
    maxIter          100;
}

"pcorr.*"
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-9;
    relTol           0;
}

p_rgh
{
    solver          PBiCGStab;
    preconditioner   DILU;
    tolerance        1e-9;
    relTol           0.01;
}

p_rghFinal
{
    $p_rgh;
    relTol           0;
}

"(U|k|epsilon).*"
{
    solver          smoothSolver;
    smoother         symGaussSeidel;
    tolerance        1e-08;
    relTol           0;
}
```

INSTITUTO D
UNIVERSIDAD DE

bria
A AMBIENTAL

Algorithm control:

- **PIMPLE** algorithm solves the pressure-velocity coupling in the Navier-Stokes equations.
- **PIMPLE** algorithm combines PISO and SIMPLE.
 - **momentumPredictor**: switch the control for solving the momentum predictor.
 - **nOuterCorrectors**: number of times the total system of equations is solved on time step.
 - **nCorrectors**: number of times the algorithm solves the pressure equation and momentum corrector in each step
 - **nOrthogonalCorrectors**: specifies repeated solutions of the pressure equation, used to update the explicit non-orthogonal correction.
 - **ddtCorr**: if set yes, reduces the the decoupling between pressure, velocity and velocity flux.

```
PIMPLE
{
    momentumPredictor    no;
    nOuterCorrectors      2;
    nCorrectors            2;
    nNonOrthogonalCorrectors 0;

    ddtCorr               yes;
    correctPhi             no;

    moveMeshOuterCorrectors no;
    turbOnFinalIterOnly    no;
}
```

Algorithm control:

- **RelaxationFactor:** controls of the under-relaxation, a technique used to for improving stability.
- **Cache:** controls data storage to make future requests faster

```
relaxationFactors
{
    fields
    {
    }
    equations
    {
        "*" 1;
    }
}

cache
{
    grad(U);
}
```

IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

- Define simulation parameters in ***system/controlDict***
 - Solver: ***overInterDyMFoam*** (incompressible two phase Flow, with optional mesh moving and mesh topology changes)
 - startTime*** (start time for the simulation), ***endTime*** (end time for the simulation), ***deltaT*** (time step of the simulation).
 - writeInterval*** (controls the timing of write output), ***purgeWrite*** (integer representing a limit on the number of time directories that are stored).
 - maxCo*** (maximun Courant Number), ***maxAlphaCo*** (maximun Courant number for the pase fields), ***maxDeltaT*** (upper limit of the time step).

```
libs          ("liboverSet.so");
application   overInterDyMFoam ;
startFrom     latestTime;
startTime     0.0;
stopAt        endTime;
endTime       20;
deltaT        0.001;
writeControl   adjustableRunTime;
writeInterval  0.1;
purgeWrite     0;
writeFormat    ascii;
writePrecision 12;
writeCompression off;
timeFormat     general;
timePrecision  6;
runTimeModifiable yes;
adjustTimeStep yes;

maxCo          2.0;
maxAlphaCo     2.0;
maxDeltaT      1;
```

- Update the runtime postprocessing sensors (*system/controlDict*)
 - to get the iso-Surface of the free surface elevation:

```
functions
{
    freeSurface
    {
        type surfaces;
        functionObjectLibs
        (
            "libsampling.so"
        );
        writeControl outputTime;
        outputInterval 1;
        surfaceFormat stl;
        interpolationScheme cellPoint;
        surfaces
        (
            topFreeSurface
            {
                type isoSurface;
                isoField alpha.water;
                isoValue 0.5;
                interpolate true;
            }
        );
        fields
        (
            alpha.water
        );
    }
}
```

IH can
INSTITUTO DE HID
UNIVERSIDAD DE CANTA

ria
BIENTAL

- Decompose case:
 - ***system/decomposeParDict***. To run the simulation in parallel, it can be decomposed using the following file :
 - ***numberOfSubdomains***: set the number of parts in which the domain is going to be splitted.
 - ***n***: it should be equal to the number of subdomains
- Run the command:
`$ decomposePar`

```
numberOfSubdomains 4;  
  
method             hierarchical;  
  
coeffs  
{  
    n                (2 2 1);  
    delta            0.001;  
    order            xyz;  
}
```


- Run the case!

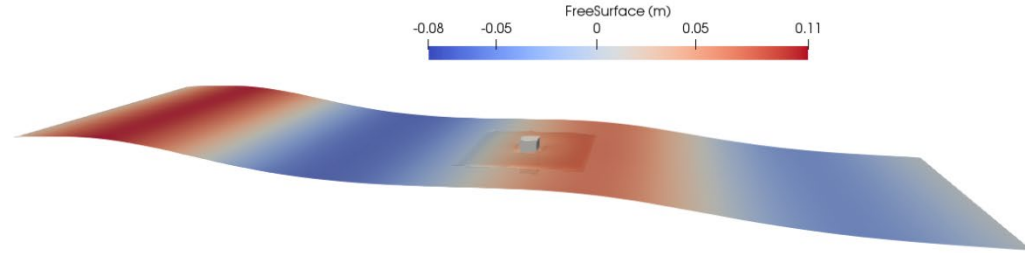
```
$ mpirun -np 4 overInterDyMFoam -parallel > log.OverWaves &
```

```
$ tail -f log.OverWaves
```

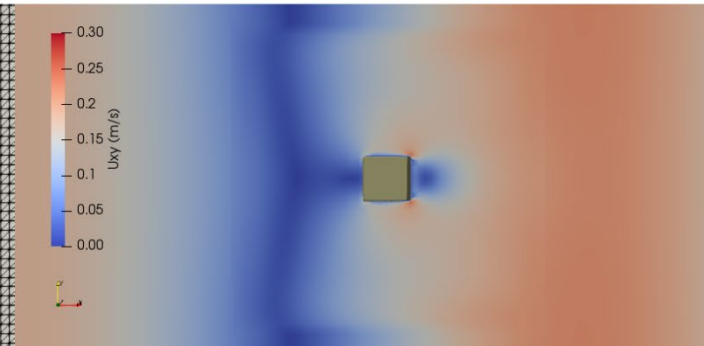
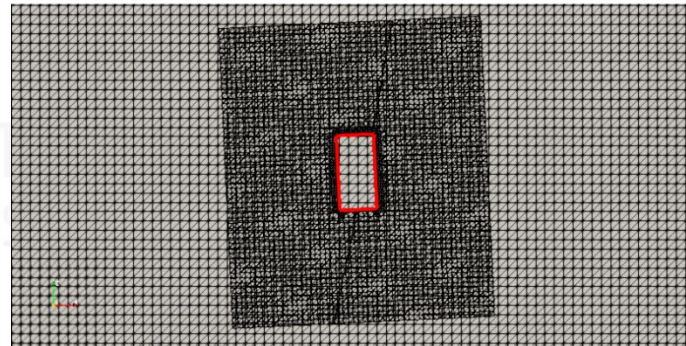
```
$ kill PID number
```

- Postprocessing with Paraview:

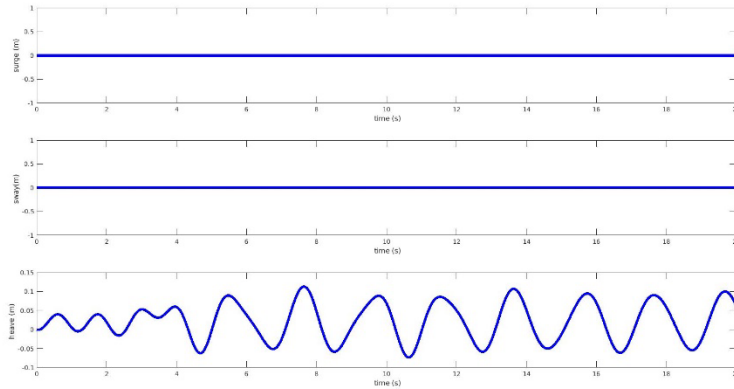
```
$ paraview
```



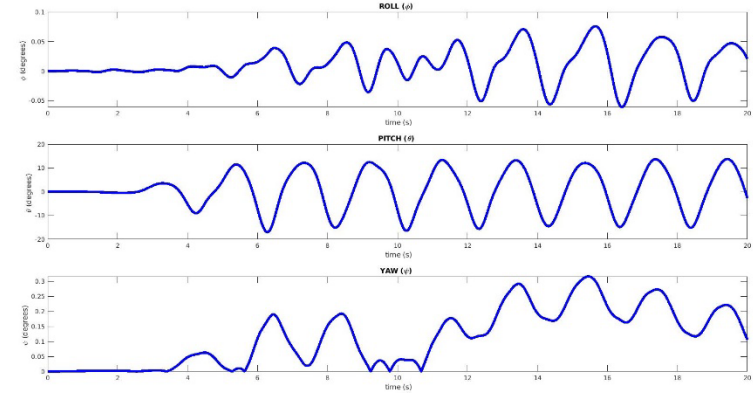
Time: 20.00 s.



- Postprocessing using Matlab (six degrees of freedom, taken from log.OverWaves):

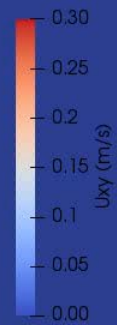
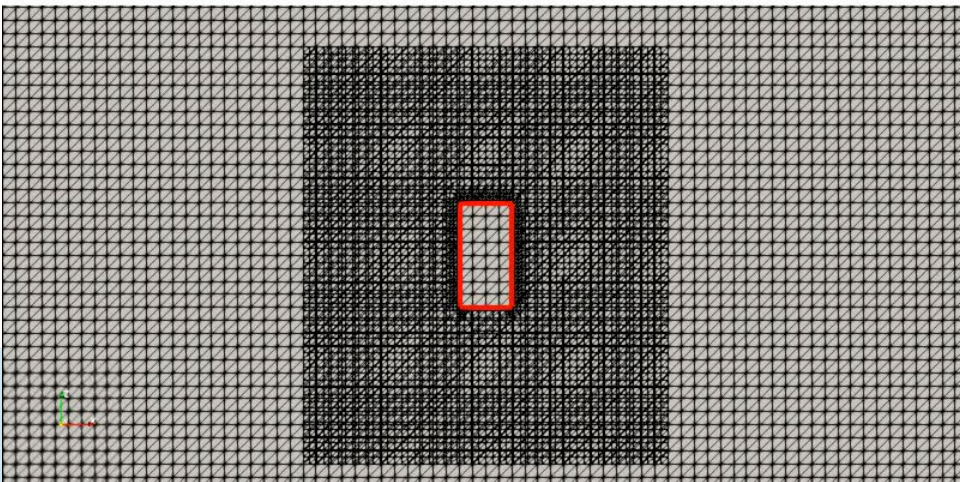
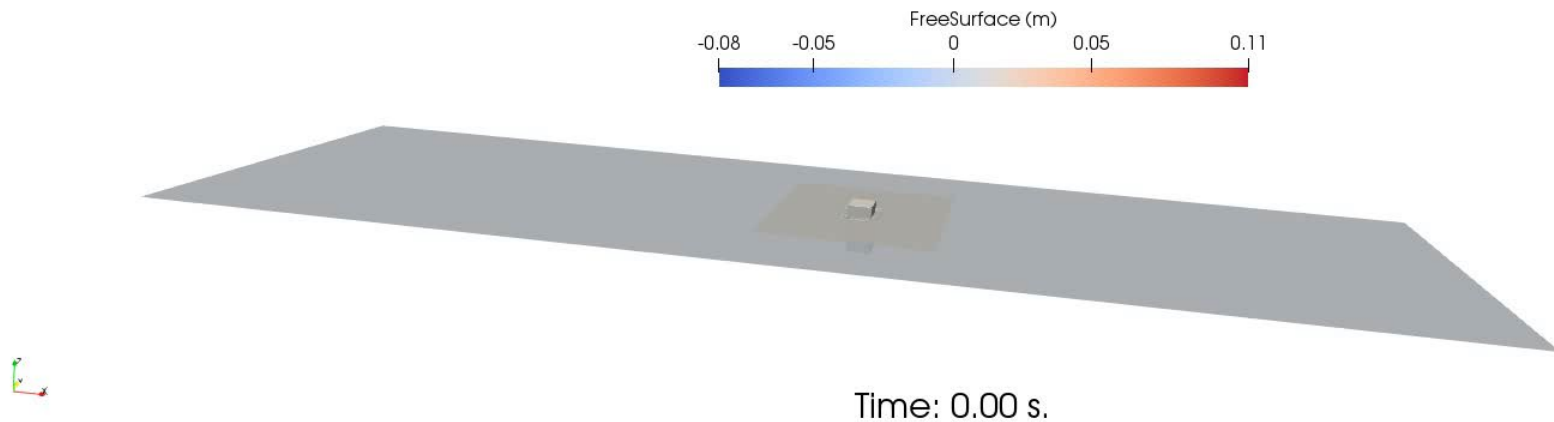


Displacements

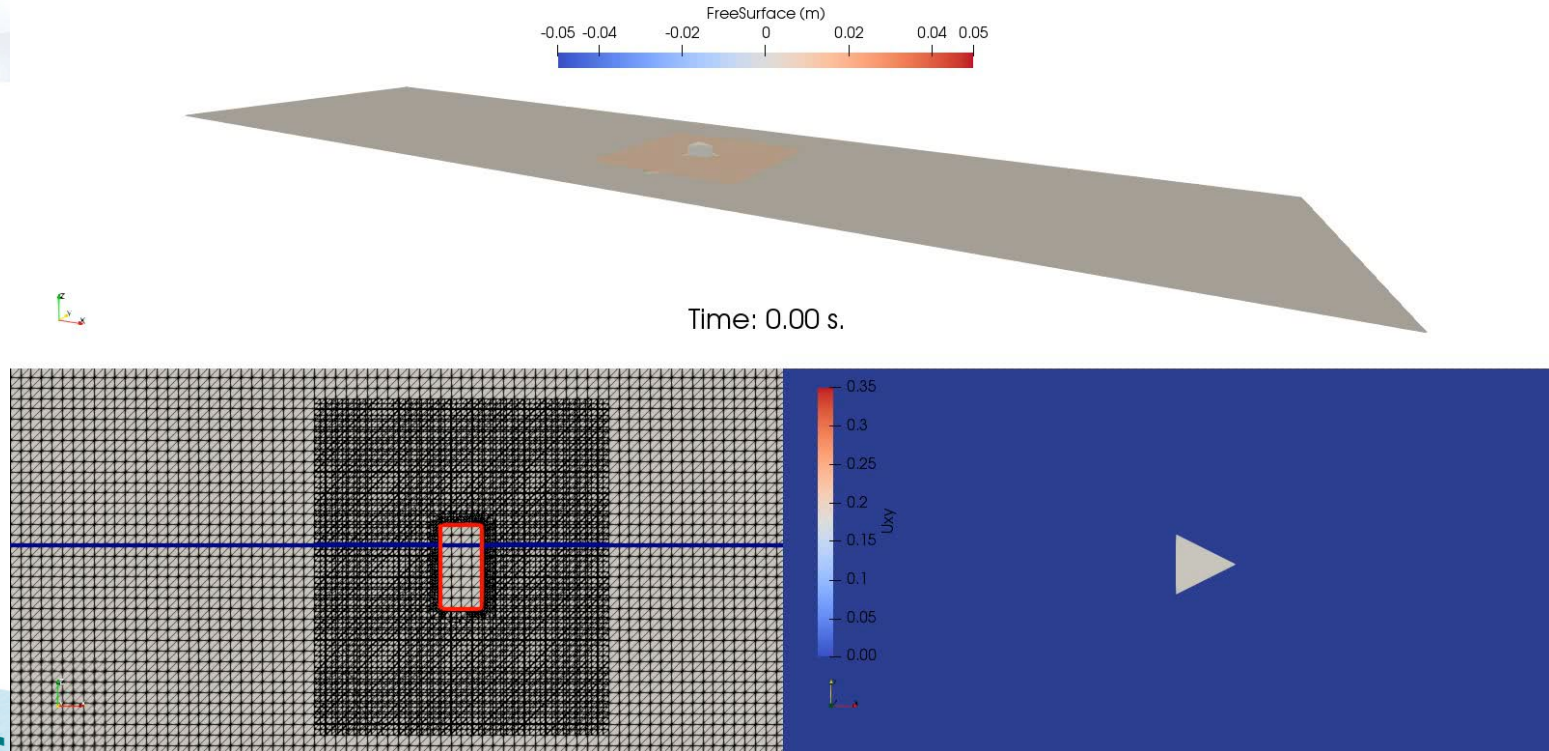


Rotations

INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

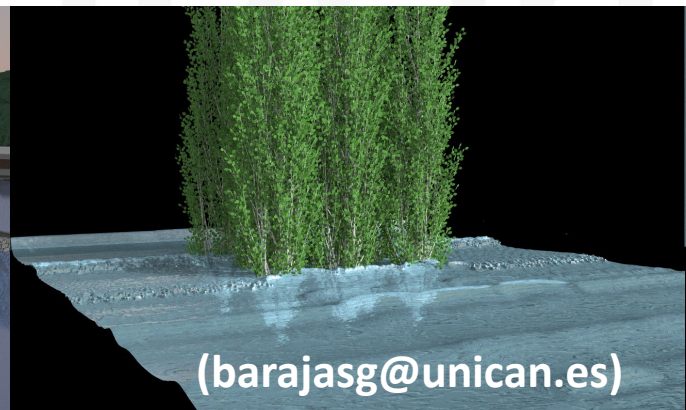
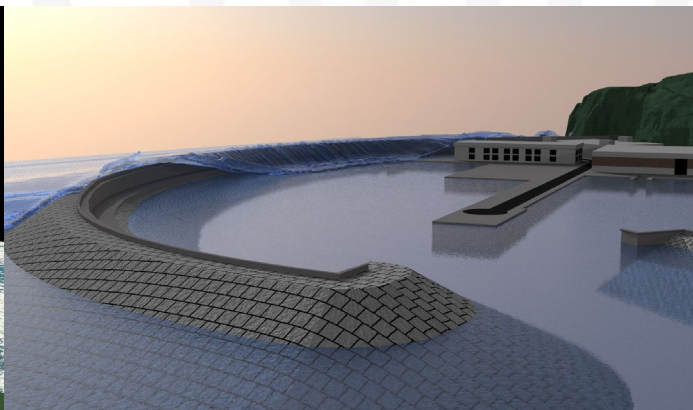
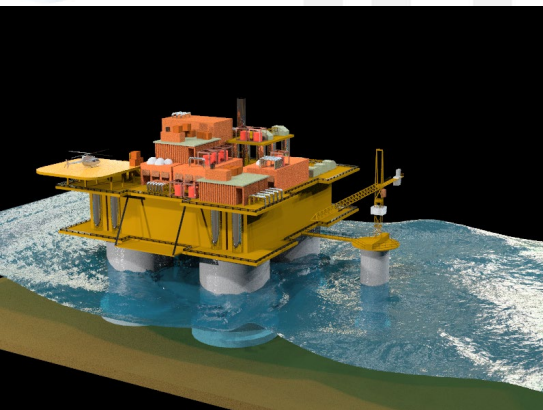


- EXERCISE: change the geometry of the floating object.





Javier L. Lara, Gabriel Barajas, María Maza, Alessandro Romano.



(barajasg@unican.es)