



ROME
2024

38TH
INTERNATIONAL
CONFERENCE
ON COASTAL
ENGINEERING

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

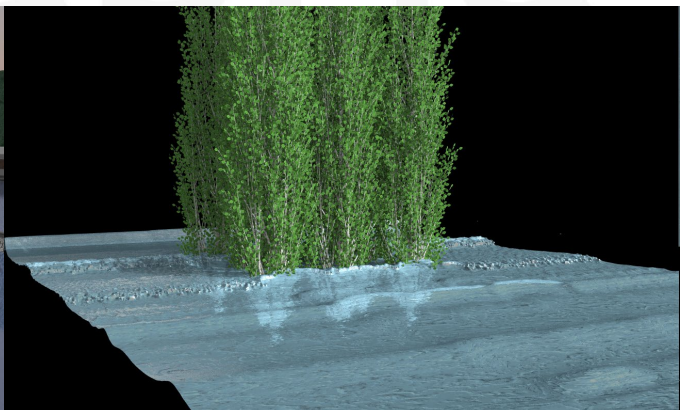
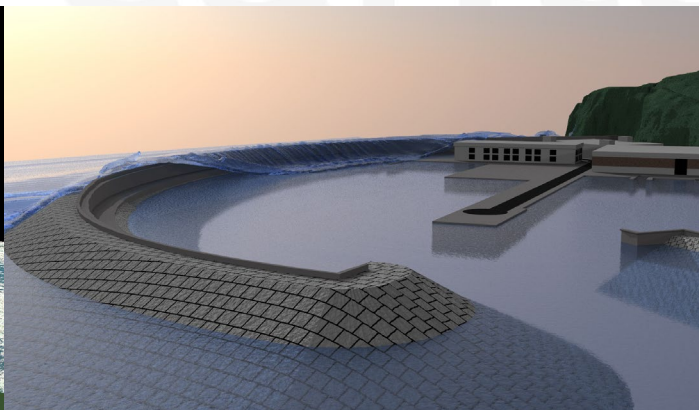
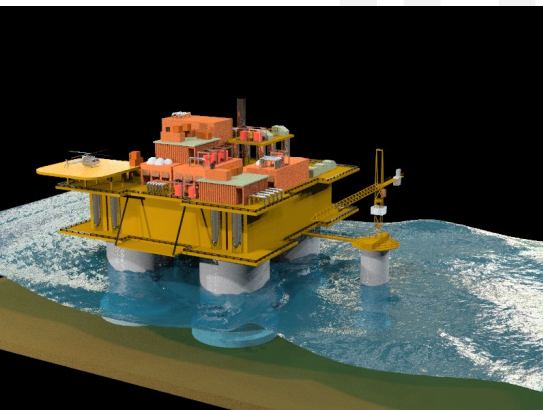


IHFOAM

applied to Coastal Engineering

Regular waves interaction with a vertical breakwater on a rubble-mound foundation

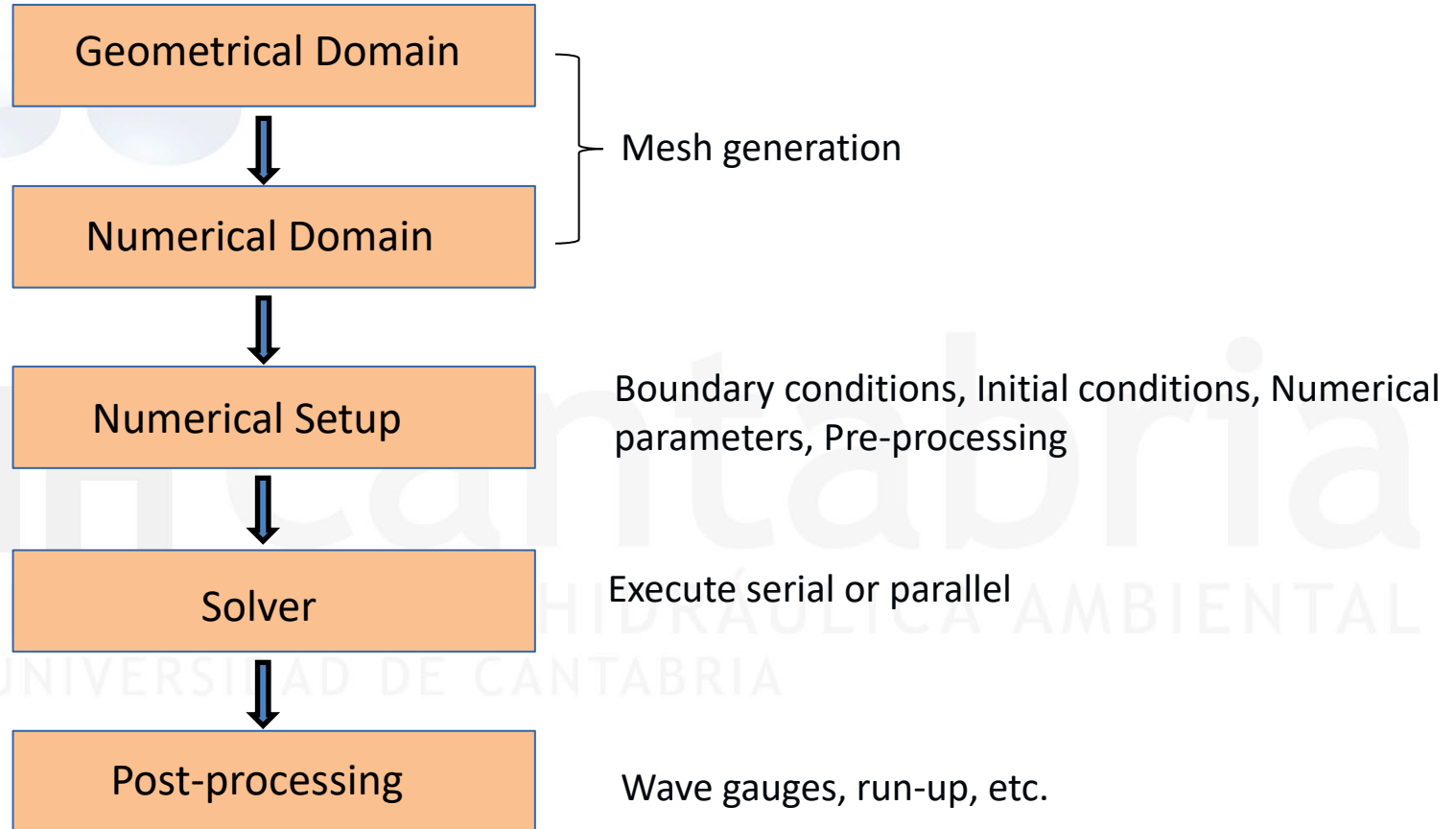
Javier L. Lara, Gabriel Barajas, María Maza, Alessandro Romano.



- We will create a 3D case from an existing 2D tutorial:
 - Create a new folder (if it does not exist yet):
`$ mkdir ~/IHFoamCourse/`
 - Copy an existing case (2D tutorial) and rename the folder:
`$ cp -r ~/OpenFOAM-v2406/tutorials/multiphase/interFoam/laminar/
waves/waveExampleStokesII ~/IHFoamCourse/.`
- Set the case:
 - Rename the case:
`$ mv ~/IHFoamCourse/waveExampleStokesII ~/IHFoamCourse/breakwater`
 - Set the OpenFOAM environment:
`$ source ~/OpenFOAM/OpenFOAM-v2406/etc/bashrc`
 - Ensure that everything you do not need is deleted:
`$ cd ~/IHFoamCourse/breakwater`
`$./Allclean`

- Download **IHFOAM** from IHCantabria's web page:
\$ <https://ihfoam.ihcantabria.com/source-download/>
- Go to the folder (it can be placed anywhere) and uncompress/extract it:
\$ cd path_to_IHFOAM_folder
\$ tar -xvzf ihFoam.tar.gz
- Set OpenFOAM environment:
\$ source ~/OpenFOAM/OpenFOAM-v2406/etc/bashrc
- First, clean any pre-compiled version:
\$ wclean
- Then, compile it:
\$ wmake

OpenFOAM workflow



OpenFOAM case

0

- alpha.water
- p_rgh
- U
- porosityIndex
- k
- epsilon
- nut

constant

- g
- transportProperties
- turbulenceProperties
- porosityDict
- waveProperties

system

- blockMeshDict
- setFieldsDict
- snappyHexMeshDict
- extrudeMeshDict
- fvSchemes
- fvSolution
- decomposeParDict
- controlDict

- Update *system/blockMeshDict*:

```
vertices
(
  ( 0.0 -20.0 0.0)
  ( 260.0 -20.0 0.0)
  ( 260.0 220.0 0.0)
  ( 0.0 220.0 0.0)
  ( 0.0 -20.0 30.0)
  ( 260.0 -20.0 30.0)
  ( 260.0 220.0 30.0)
  ( 0.0 220.0 30.0)
);

blocks
(
  hex (0 1 2 3 4 5 6 7) (130 120 30) simpleGrading (1 1 1)
);
```

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

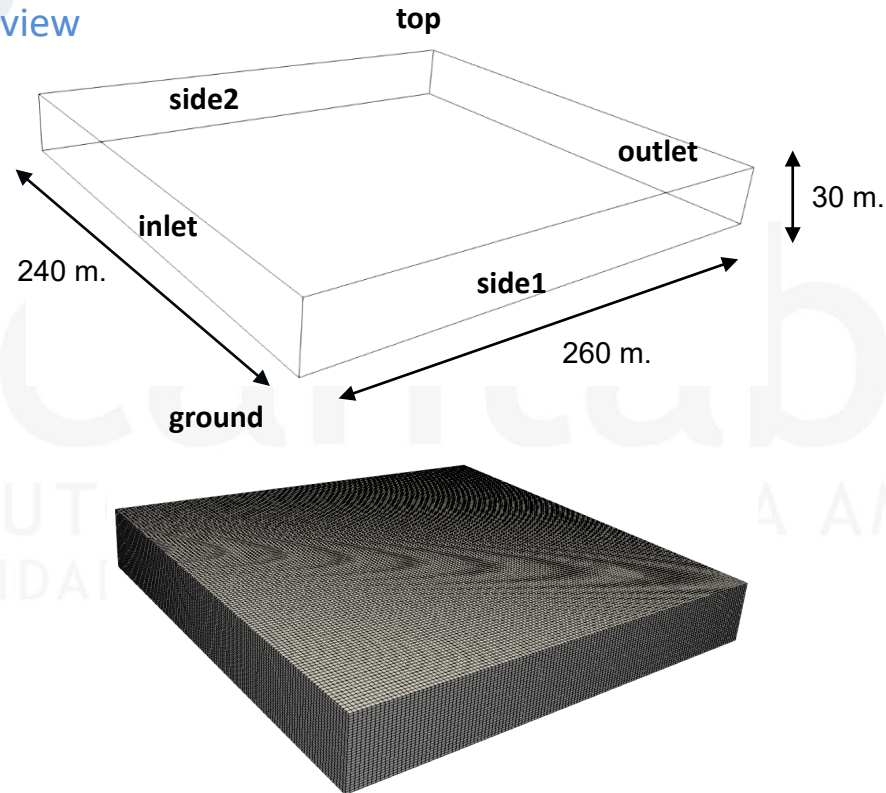
```
boundary
(
  inlet
  {
    type patch;
    faces
    (
      (0 4 7 3)
    );
  }
  outlet
  {
    type patch;
    faces
    (
      (1 5 6 2)
    );
  }
  ground
  {
    type wall;
    faces
    (
      (0 1 2 3)
    );
  }
  top
  {
    type patch;
    faces
    (
      (4 5 6 7)
    );
  }
  side1
  {
    type patch;
    faces
    (
      (0 1 5 4)
    );
  }
  side2
  {
    type patch;
    faces
    (
      (3 2 6 7)
    );
  }
);
```

Regular waves interaction with a vertical breakwater on a rubble-mound foundation

IH cantabria
INSTITUTO DE
HIDRAULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

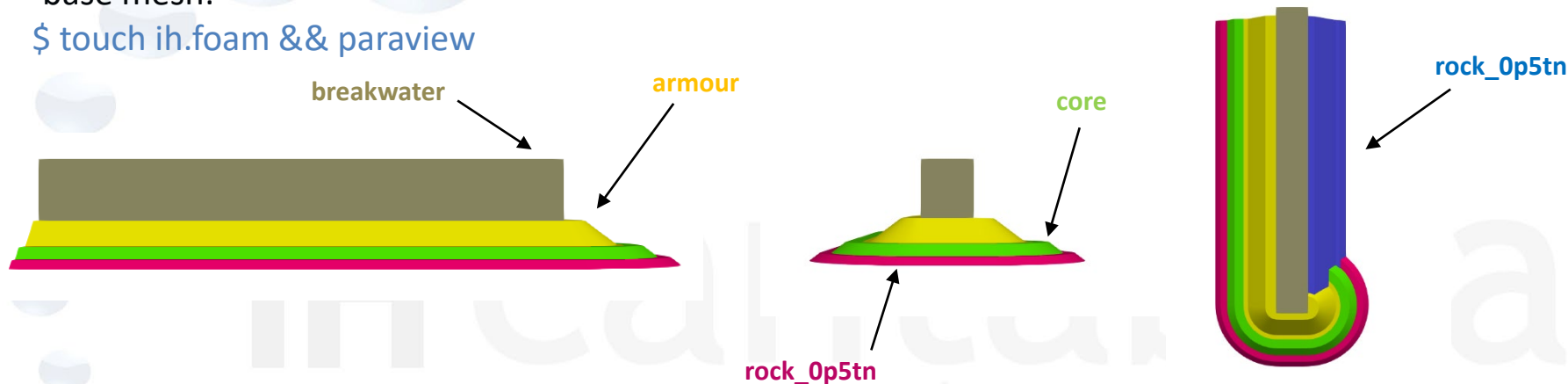
- Display the base mesh using Paraview:

\$ touch ih.foam && paraview



- Define and create the rubble mound elements (using Autocad, Rhino, etc.).
- Check the .stl files; open Paraview, load the .stl files (5) and check that the geometries fits the base mesh:

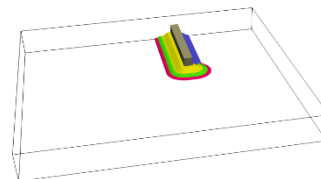
\$ touch ih.foam && paraview



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

\$ mkdir constant/triSurface

\$ cp *.stl constant/triSurface/.



- Use **snappyHexMesh** as mesh generator to take the existing base mesh and re-mesh it to fit the real geometry.
- Copy snappyHexMeshDict from a tutorial:
`$ cp -r ~/OpenFOAM-v2406/tutorials/multiphase/interFoam/RAS/mixerVesselAMI/system/
snappyHexMeshDict ~/IHFoamCourse/wavesCylinders/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
{
  breakwater.stl
  {
    type triSurfaceMesh;
    name breakwater;
  }
};
```

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

```
nCellsBetweenLevels 3;
```

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

```
features
(
);
```

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

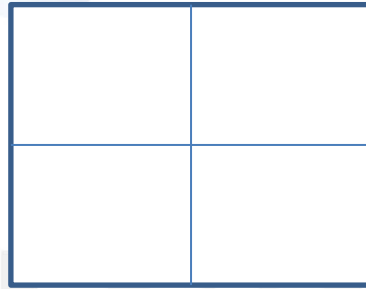
```
refinementSurfaces
{
  breakwater
  {
    level (2 2);
  }
}
```

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

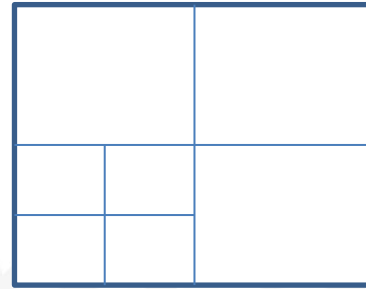
- Refinement levels in OpenFOAM: increase in the refinement level reduces the cell size by half.



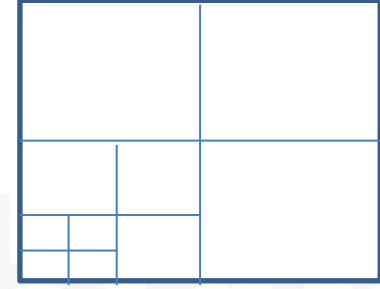
Level 0



Level 1



Level 2



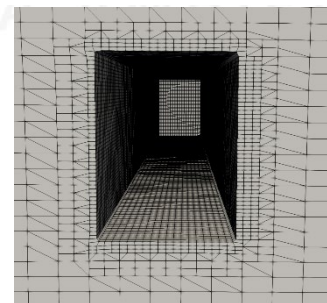
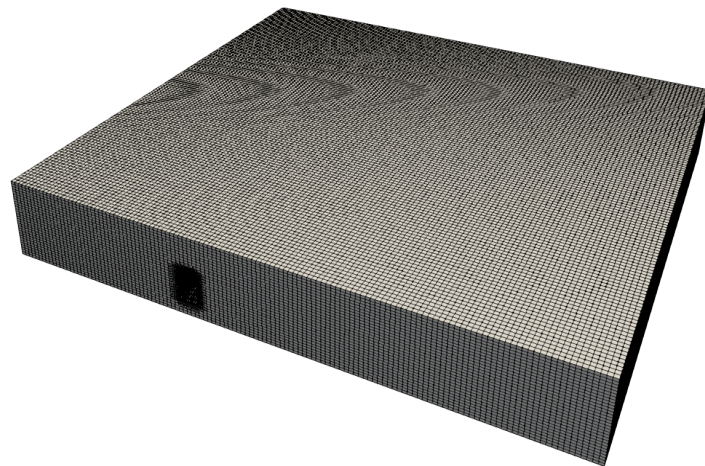
Level 3

IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

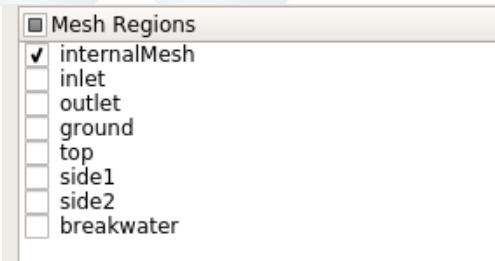
```
locationInMesh (30 10 5.0);
```

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

- Create the intermediate mesh:
\$ `snappyHexMesh -overwrite`.
- Check the intermediate mesh quality:
\$ `checkMesh`
- Check the intermediate 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.)



- 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.



```

7
(
  inlet
  {
    type            patch;
    nFaces          3600;
    startFace       1571187;
  }
  outlet
  {
    type            patch;
    nFaces          3600;
    startFace       1574787;
  }
  ground
  {
    type            wall;
    inGroups        1(wall);
    nFaces          15600;
    startFace       1578387;
  }
  top
  {
    type            patch;
    nFaces          15600;
    startFace       1593987;
  }
)

```

```

side1
{
  type            patch;
  nFaces          3900;
  startFace       1609587;
}
side2
{
  type            patch;
  nFaces          4239;
  startFace       1613487;
}
breakwater
{
  type            wall;
  inGroups        1(wall);
  nFaces          23568;
  startFace       1617726;
}
)

```


- Once the final boundaries are known, update and add the field values in the **0.org** folder:

– alpha.water:

```
sides
{
  type      empty;
}
```



```
side1
{
  type      zeroGradient;
}
side2
{
  type      zeroGradient;
}
```

```
breakwater
{
  type      zeroGradient;
}
```

– p_rgh:

```
sides
{
  type      empty;
}
```



```
side1
{
  type      fixedFluxPressure;
  value     uniform 0;
}
side2
{
  type      fixedFluxPressure;
  value     uniform 0;
}
```

```
breakwater
{
  type      fixedFluxPressure;
  value     uniform 0;
}
```

— U:

```
sides
{
  type      empty;
}
```



```
side1
{
  type      slip;
}
side2
{
  type      slip;
}
```

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

IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

- 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:

INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

k

```

dimensions      [0 2 -2 0 0 0];
internalField   uniform 0.06;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      kqRWallFunction;
        value      uniform 0.06;
    }
    side1
    {
        type      zeroGradient;
    }
    side2
    {
        type      zeroGradient;
    }
    top
    {
        type      inletOutlet;
        inletValue uniform 0.06;
        value      uniform 0.06;
    }
    breakwater
    {
        type      kqRWallFunction;
        value      uniform 0.06;
    }
}

```

epsilon

```

dimensions      [0 2 -3 0 0 0];
internalField   uniform 0.00075;

boundaryField
{
    inlet
    {
        type      zeroGradient;
    }
    outlet
    {
        type      zeroGradient;
    }
    ground
    {
        type      epsilonWallFunction;
        value      uniform 0.00075;
    }
    side1
    {
        type      zeroGradient;
    }
    side2
    {
        type      zeroGradient;
    }
    top
    {
        type      inletOutlet;
        inletValue uniform 0.00075;
        value      uniform 0.00075;
    }
    breakwater
    {
        type      epsilonWallFunction;
        value      uniform 0.00075;
    }
}

```

nut

```

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;
        value      uniform 0;
    }
    side1
    {
        type      calculated;
        value      uniform 0;
    }
    side2
    {
        type      calculated;
        value      uniform 0;
    }
    top
    {
        type      calculated;
        value      uniform 0;
    }
    breakwater
    {
        type      nutkWallFunction;
        value      uniform 0;
    }
}

```

- As the case is defined with several porous zones, a porosity variable (***porosityIndex***) must be defined and added to the **0.org** folder:

\$ touch 0.org/porosityIndex

```
FoamFile
{
  version      2.0;
  format       ascii;
  class        volScalarField;
  location     "0";
  object       porosityIndex;
}
// * * * * *

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

```
boundaryField
{
  inlet
  {
    type          zeroGradient;
  }
  outlet
  {
    type          zeroGradient;
  }
  ground
  {
    type          zeroGradient;
  }
  side1
  {
    type          zeroGradient;
  }
  side2
  {
    type          zeroGradient;
  }
  top
  {
    type          zeroGradient;
  }
  breakwater
  {
    type          zeroGradient;
  }
}
```

INSTITUTO
UNIVERSIDAD D

bria
A AMBIENTAL

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

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
    volScalarFieldValue porosityIndex 0
);

regions
(
    boxToCell
    {
        box (-100.0 -100.0 0.0) (500.0 500.0 13.0);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }

    surfaceToCell
    {
        file                "./constant/triSurface/armour.stl";
        outsidePoints        ((10.0 10.0 10.0));
        includeCut           true;
        includeInside        true;
        includeOutside       false;
        nearDistance         -1;
        curvature            -100;
        fieldValues
        (
            volScalarFieldValue porosityIndex 1
        );
    }
}
```

Set initial water depth

Define *armour* zone as porous.

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

```
surfaceToCell
{
    file                "./constant/triSurface/core.stl";
    outsidePoints        ((10.0 10.0 10.0));
    includeCut           true;
    includeInside         true;
    includeOutside        false;
    nearDistance          -1;
    curvature              -100;
    fieldValues
    (
        volScalarFieldValue porosityIndex 2
    );
}
surfaceToCell
{
    file                "./constant/triSurface/rock_0p5tn.stl";
    outsidePoints        ((10.0 10.0 10.0));
    includeCut           true;
    includeInside         true;
    includeOutside        false;
    nearDistance          -1;
    curvature              -100;
    fieldValues
    (
        volScalarFieldValue porosityIndex 3
    );
}
```

Define *core* zone as porous.

Define *rock_0p5tn* zone as porous.

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

```
surfaceToCell
{
    file            "./constant/triSurface/rock_1tn_to_3tn.stl";
    outsidePoints   ((10.0 10.0 10.0));
    includeCut      true;
    includeInside   true;
    includeOutside  false;
    nearDistance    -1;
    curvature       -100;
    fieldValues
    (
        volScalarFieldValue porosityIndex 4
    );
}
```

Define *rock_1tn_to_3tn* zone
as porous.

IH cantabria
INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

- A dictionary file must be added to characterized each zone following the Forcheimer formulation:
\$ touch constant/porosityDict

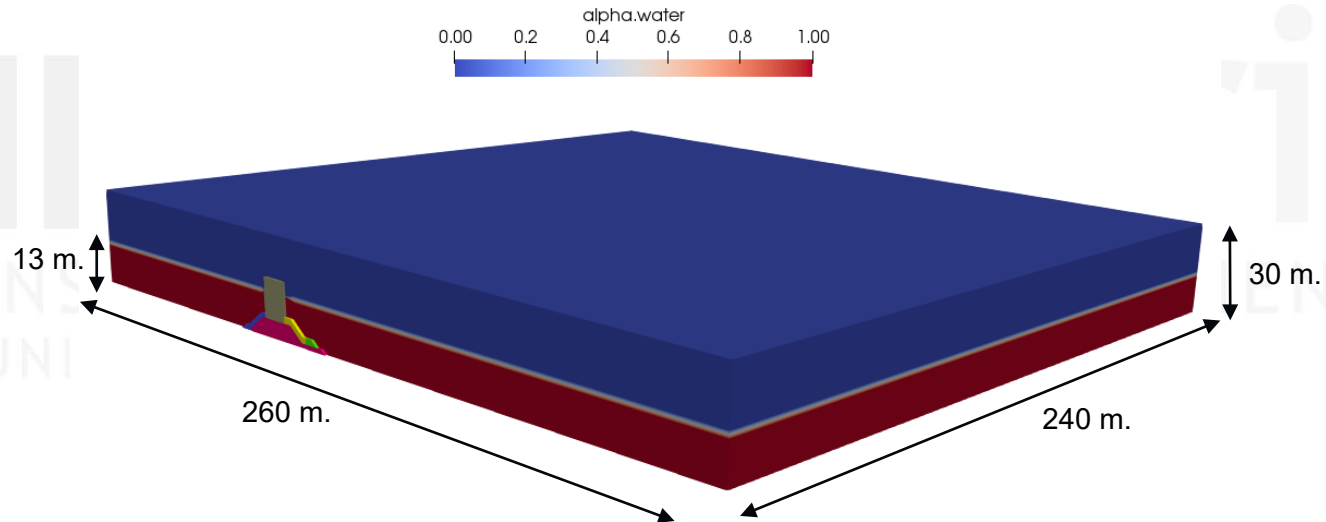
```
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location      "constant";
    object        porosityDict;
}
// *****

a          5(0 200 200 200 200);
b          5(0 1.1 0.8 1.0 1.1);
c          5(0 0.34 0.34 0.34 0.34);

D50        5(1 2.0 0.30 0.40 1.0);
porosity    5(1 0.45 0.2 0.25 0.35);
```

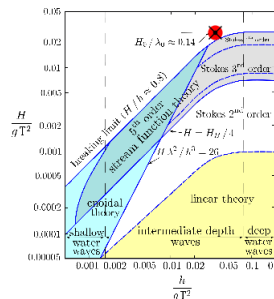
IH cantabria
INSTITUTO DE HIDRAULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

- \$ cp -r 0.org 0
\$ setFields
- Open Paraview and plot the initial set-up to ensure everything is correct:
\$ paraview



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

- **WaveModel**: Stokes V regular waves
- **nPaddle**: 16 paddles
- **waveHeight**: $H = 5.0$ m.
- **waveAngle**: 0 (in degrees)
- **rampTime**: 12.0 s (smoothing time)
- **activeAbsorption**: yes (activated absorption at generation).
- **wavePeriod**: $T = 6.0$ s.
- Define 8 paddles to absorb at the outlet.



```
inlet
{
    alpha            alpha.water;

    waveModel         StokesII;

    nPaddle           16;

    waveHeight        5.0;

    waveAngle         0.0;

    rampTime          12.0;

    activeAbsorption  yes;

    wavePeriod        6.0;
}
```

```
outlet
{
    alpha            alpha.water;

    waveModel         shallowWaterAbsorption;

    nPaddle           8;
}
```

- 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;
}
  
```


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 linearUpwind grad(U);
    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 linearUpwind grad(U)*** is second order, ***Gauss vanLeer*** is second order and ***Gauss upwind*** is a first order numerical scheme)

Numerical Schemes:

```
laplacianSchemes
{
  default      Gauss linear limited 0.5;
}

interpolationSchemes
{
  default      linear;
}

snGradSchemes
{
  default      limited 0.5;
}
```

Laplacian terms, such as the diffusion term in the momentum equation (**Gauss linear** is second order discretisation scheme)

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

Component of gradient normal to a cell face.

INSTITUTO DE HIDRÁULICA AMBIENTAL
UNIVERSIDAD DE CANTABRIA

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.
 - **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.

```
"alpha.water.*"  
{  
    nAlphaCorr      1;  
    nAlphaSubCycles 2;  
    alphaOuterCorrectors yes;  
    cAlpha          1;  
  
    MULESCorr       no;  
    nLimiterIter    3;  
  
    solver           smoothSolver;  
    smoother         symGaussSeidel;  
    tolerance        1e-8;  
    relTol           0;  
}
```

Algorithm control:

- For each variable solved in the particular equation, the type of solver and parameters 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.

```
pcorr
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.1;
}
pcorrFinal
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-7;
    relTol           0;
}
p_rgh
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-6;
    relTol           0.1;
}
p_rghFinal
{
    solver          PCG;
    preconditioner   DIC;
    tolerance        1e-7;
    relTol           0;
}
"(U|k|epsilon)"
{
    solver          PBiCG;
    preconditioner   DILU;
    tolerance        1e-6;
    relTol           0.1;
}
"(U|k|epsilon)Final"
{
    solver          PBiCG;
    preconditioner   DILU;
    tolerance        1e-7;
    relTol           0;
}
}
```

Regular waves interaction with a vertical breakwater on a rubble-mound foundation

INSTITUTO DE
UNIVERSIDAD DE

abria
LICA 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.
 - **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..

```
PIMPLE
{
    momentumPredictor no;
    nCorrectors          3;
    nNonOrthogonalCorrectors 0;
    nAlphaCorr           1;
    nAlphaSubCycles      4;
    cAlpha                1;
}
```

- Define simulation parameters in ***system/controlDict***
 - Solver: ***interFoam*** (incompressible two phase flow)
 - 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*** (maximum Courant Number), ***maxAlphaCo*** (maximum Courant number for the phase fields), ***maxDeltaT*** (upper limit of the time step).

Regular waves interaction with a vertical breakwater on a rubble-mound foundation

```

application      ihFoam;

startFrom        latestTime;

startTime        0;

stopAt           endTime;

endTime          150.0;

deltaT           0.01;

writeControl     adjustableRunTime;

writeInterval    0.1;

purgeWrite       0;

writeFormat      ascii;

writePrecision   6;

writeCompression off;

timeFormat       general;

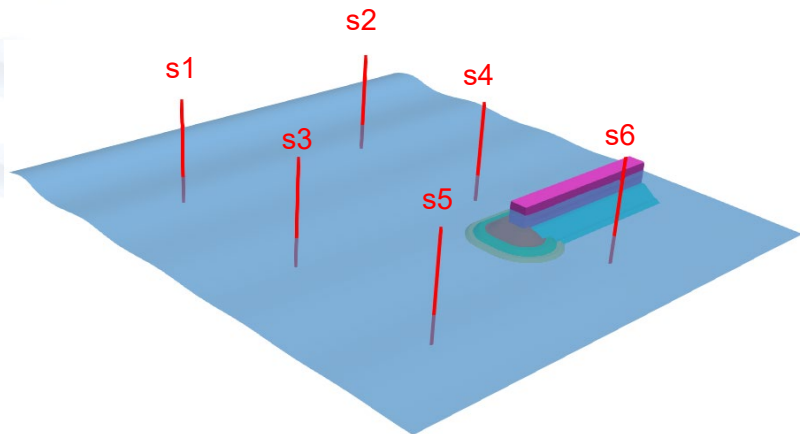
timePrecision    6;

runTimeModifiable yes;

adjustTimeStep   yes;

maxCo            0.65;
maxAlphaCo       0.65;
maxDeltaT        0.05;
  
```


- Update the runtime postprocessing sensors (*system/controlDict*)
 - to get the free surface at some specific positions along the domain (define 6 gauges).



```
line
{
    type            sets;
    functionObjectLibs ("libsampling.so");
    enabled          true;
    writeControl      outputTime;
    outputInterval    1;

    interpolationScheme cellPoint;
    setFormat         raw;
    sets
    (
        line1
        {
            type uniform;
            axis distance;
            start ( 50.0 40.0 0.0 );
            end   ( 50.0 40.0 30.0 );
            nPoints 1001;
        }
        line2
        {
            type uniform;
            axis distance;
            start ( 50.0 150.0 0.0 );
            end   ( 50.0 150.0 30.0 );
            nPoints 1001;
        }
        line3
        {
            type uniform;
            axis distance;
            start ( 130.0 40.0 0.0 );
            end   ( 130.0 40.0 30.0 );
            nPoints 1001;
        }
        line4
        {
            type uniform;
            axis distance;
            start ( 130.0 150.0 0.0 );
            end   ( 130.0 150.0 30.0 );
            nPoints 1001;
        }
    )
}
```

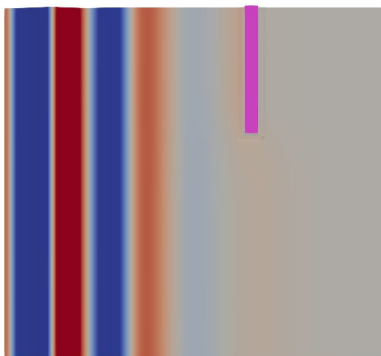
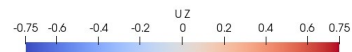
```
line5
{
    type uniform;
    axis distance;
    start ( 210.0 40.0 0.0 );
    end   ( 210.0 40.0 30.0 );
    nPoints 1001;
}
line6
{
    type uniform;
    axis distance;
    start ( 210.0 150.0 0.0 );
    end   ( 210.0 150.0 30.0 );
    nPoints 1001;
}
line7
{
    type uniform;
    axis distance;
    start ( 30.0 1.01 0.0 );
    end   ( 30.0 1.01 60.0 );
    nPoints 1001;
}
line8
{
    type uniform;
    axis distance;
    start ( 60.0 1.01 0.0 );
    end   ( 60.0 1.01 60.0 );
    nPoints 1001;
}

);
fixedLocations false;
fields
(
    alpha.water
);
}
```

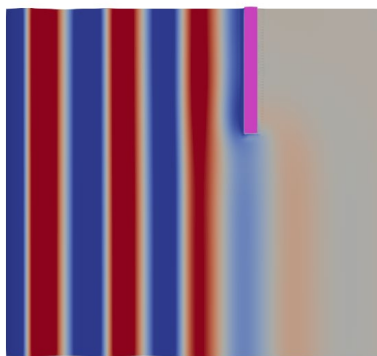
- Decompose case:
 - ***system/decomposeParDict***. To run the simulation in parallel, it can be decomposed using the following file:
 - ***numberOfSubdomains***: set the number of parts in which the domain is going to be splitted.
 - ***n***: it should be equal to the number of subdomains
- Run the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

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

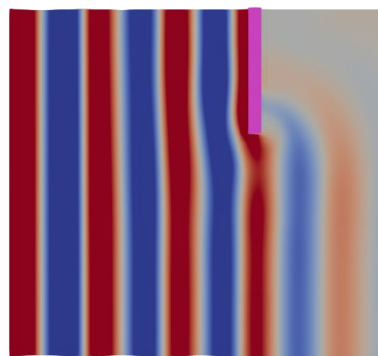
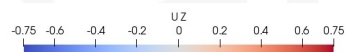
- Run the case!
\$ `mpirun -np 2 ihFoam -parallel > log.breakwater &`
\$ `tail -f log.breakwater`
\$ kill PID number
- Postprocessing with Paraview:
\$ `paraview`



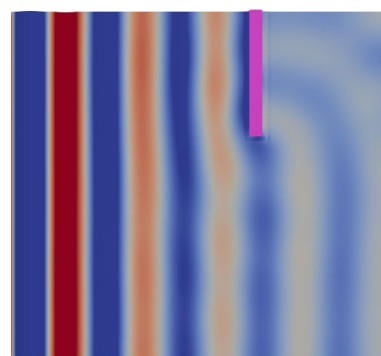
Time: 20.00 s.



Time: 30.00 s.

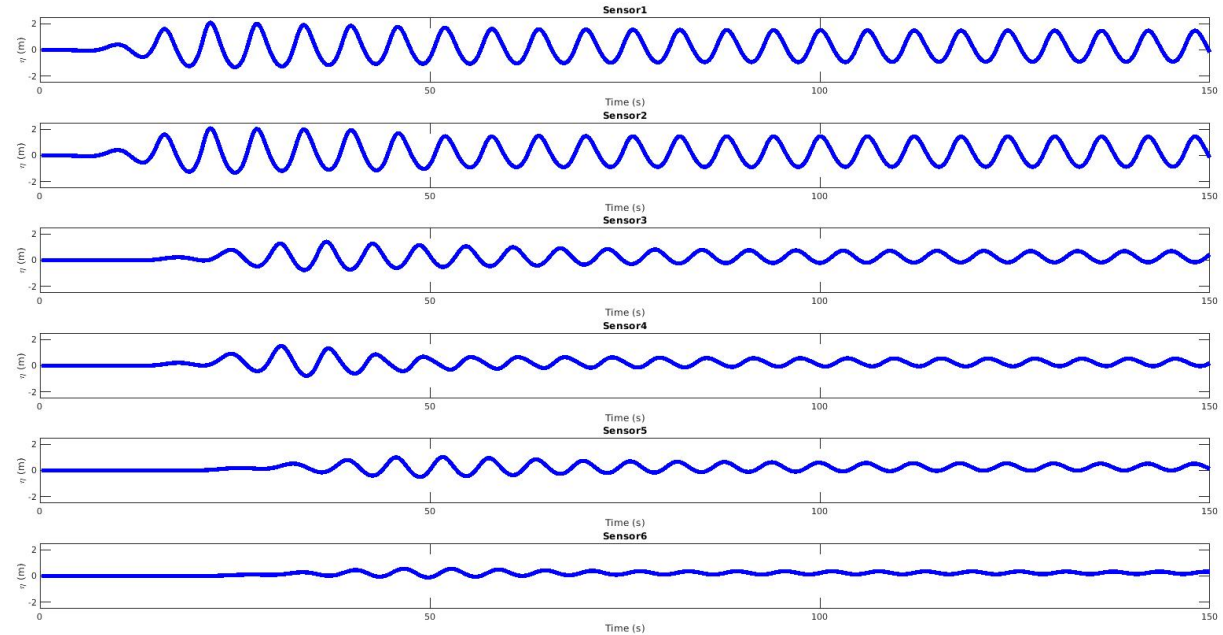
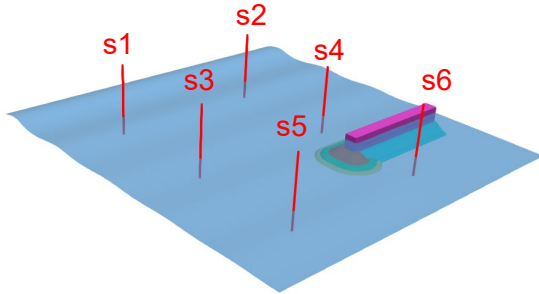


Time: 40.00 s.

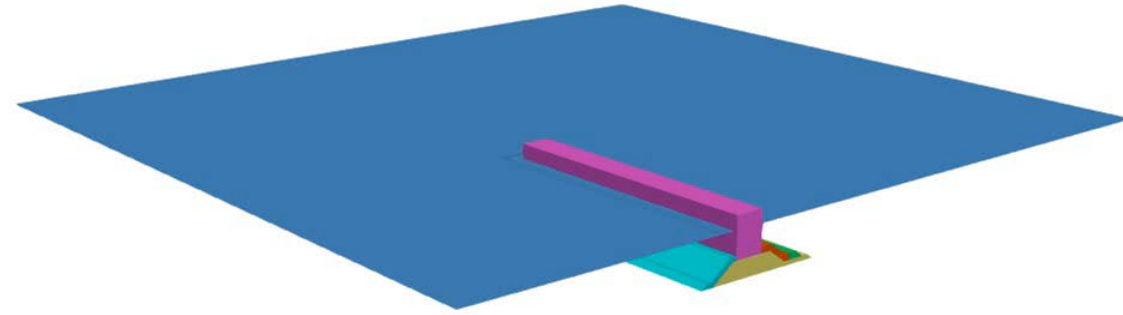


Time: 150.00 s.

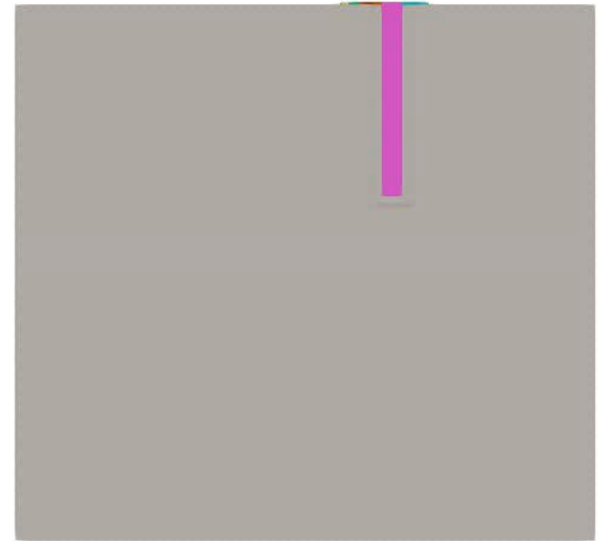
- Postprocessing using Matlab (plot of the free-surface gauges, taken from the ***postProcessing*** folder):



Regular waves interaction with a vertical breakwater on a rubble-mound foundation

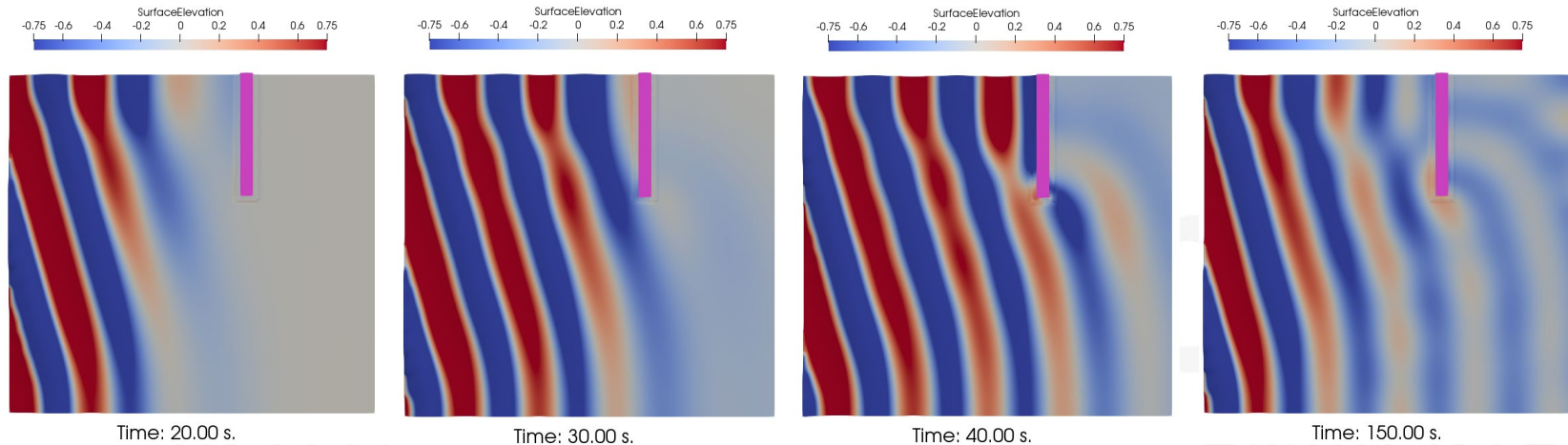


Time: 0.00 s.



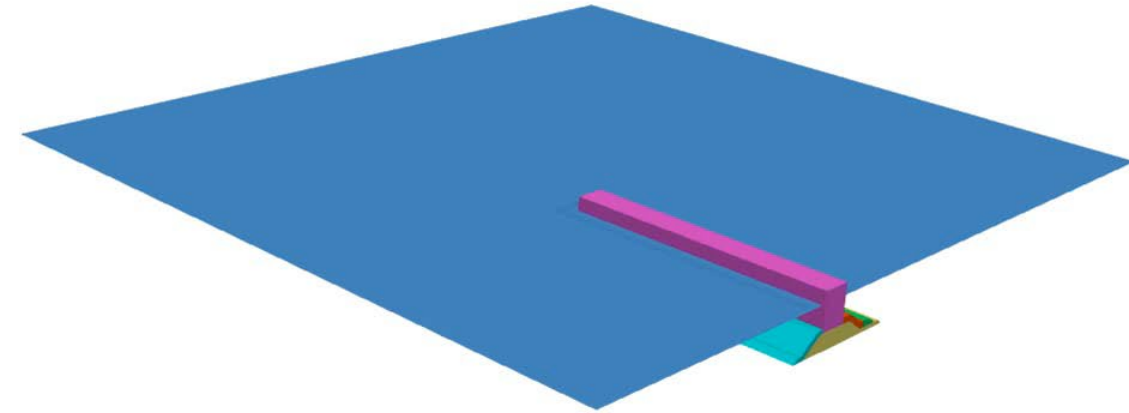
Time: 0.00 s.

- EXERCISE: Simulate directional regular waves.

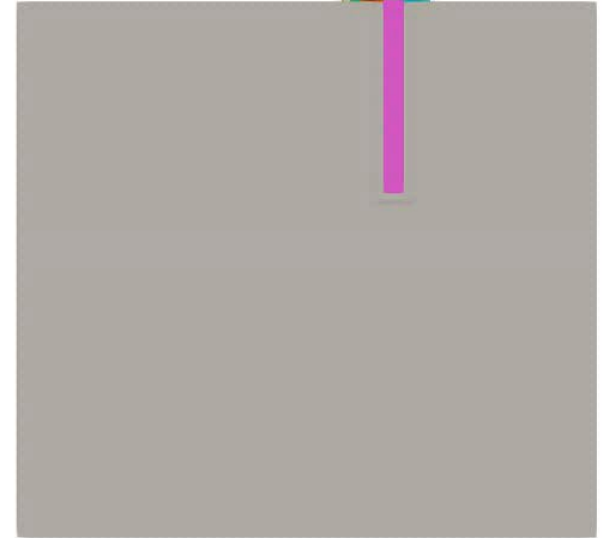


UNIVERSIDAD DE CANTABRIA

Regular waves interaction with a vertical breakwater on a rubble-mound foundation



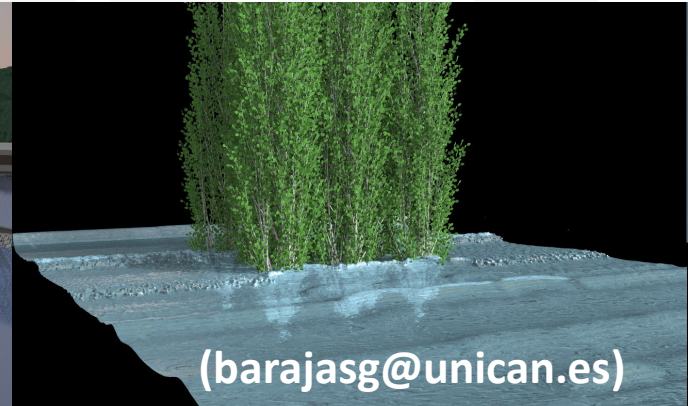
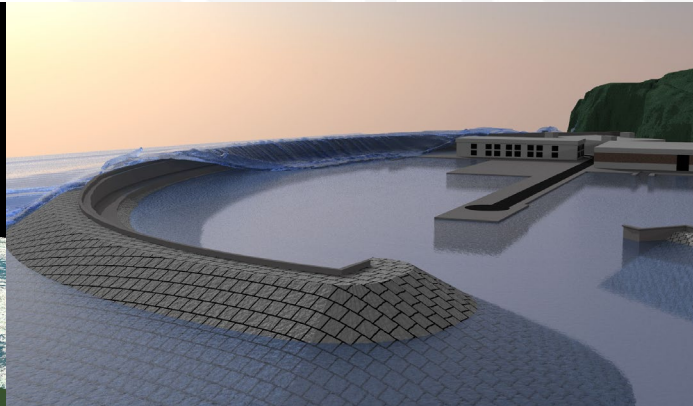
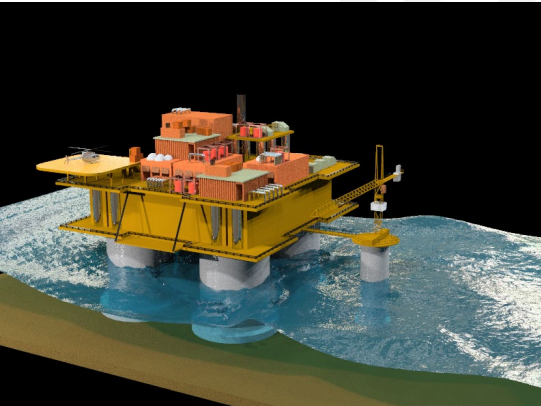
Time: 0.00 s.



Time: 0.00 s.



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



(barajasg@unican.es)