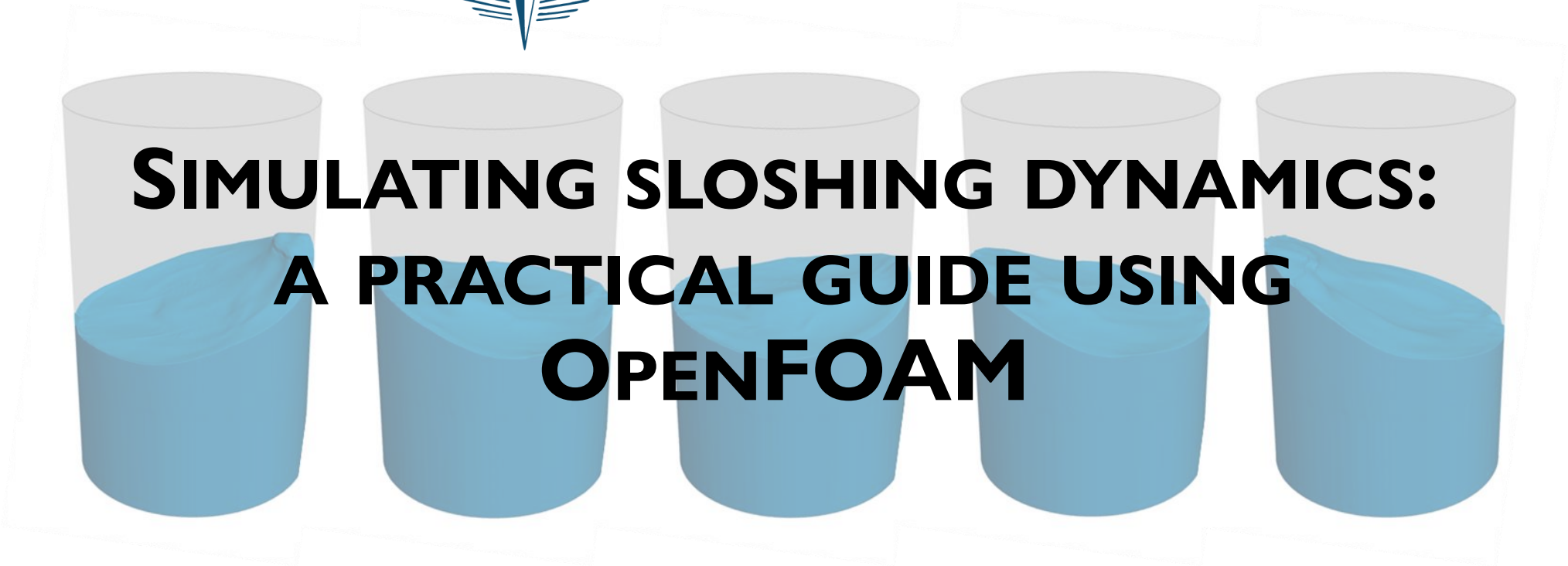
von KARMAN INSTITUTE
FOR FLUID DYNAMICS



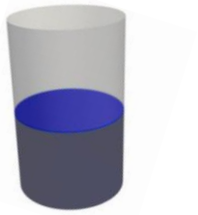
SIMULATING SLOSHING DYNAMICS: A PRACTICAL GUIDE USING OPENFOAM

Antonio Cantiani

18th March 2024

SLOSHING

"Sloshing is the dynamic motion of a liquid inside a container, typically induced by the container motion"



Relevant for multiple applications:

Sloshing dynamics

Automotive

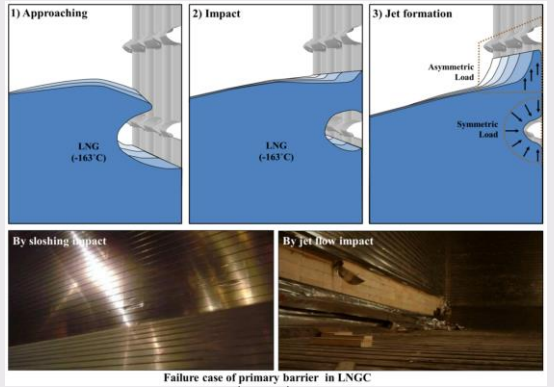
https://www.youtube.com/watch?app=desktop&v=TBohaESj_tE

Nuclear

doi.org/10.20290/estubtdb.501177

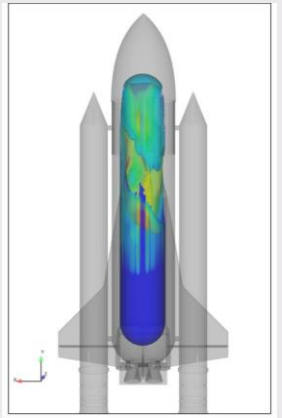
Sloshing dynamics/thermodynamics

Maritime



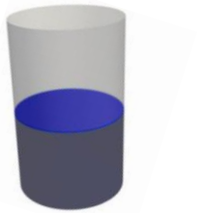
Failure case of primary barrier in LNGC
doi.org/10.3390/app9081558

Aerospace



https://www.youtube.com/watch?v=KFdg0xqfVTk

SLOSHING



"Sloshing is the dynamic motion of a liquid inside a container, typically induced by the container motion"

Relevant for multiple applications:

Sloshing dynamics

Automotive

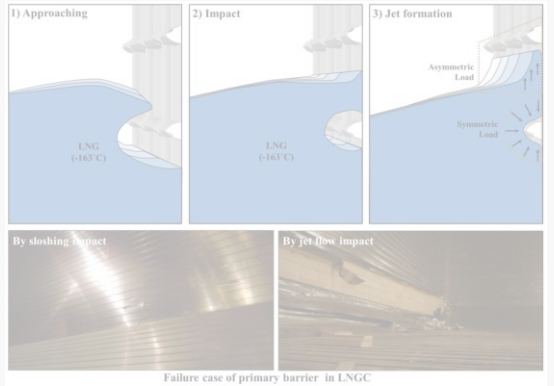
https://www.youtube.com/watch?app=desktop&v=TBohaESj_tE

Nuclear

doi.org/10.20290/estubtdb.501177

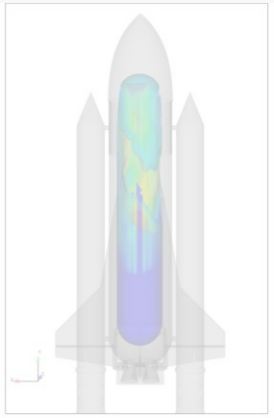
Sloshing dynamics/thermodynamics

Maritime



Failure case of primary barrier in LNGC
doi.org/10.3390/app9081558

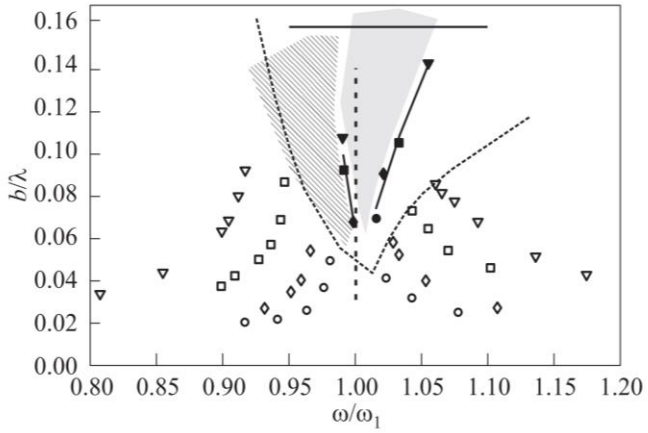
Aerospace



https://www.youtube.com/watch?v=KFdg0xqfVTk

SLOSHING DYNAMICS: MODES

Sloshing behavior depends on **forcing amplitude** and **forcing frequency**

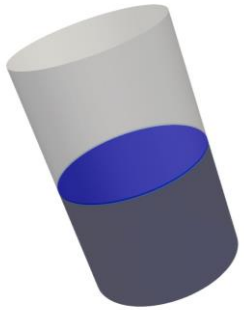


Amplitude-frequency diagram for four different forcing amplitudes [1]

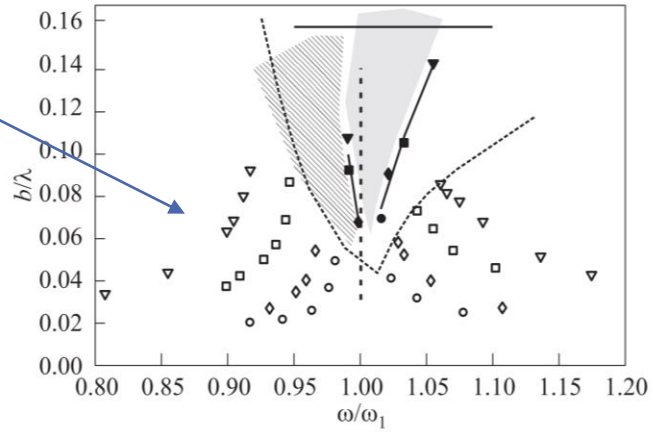
[1] Hopfinger, E. J., & Baumbach, V. (2009). Liquid sloshing in cylindrical fuel tanks. Progress in Propulsion Physics, 1, 279-292. doi: 10.1051/eucass/200901279

SLOSHING DYNAMICS: MODES

Sloshing behavior depends on **forcing amplitude** and **forcing frequency**



Linear

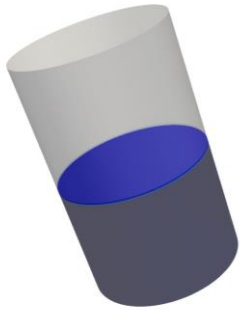


Amplitude-frequency diagram for four different forcing amplitudes [1]

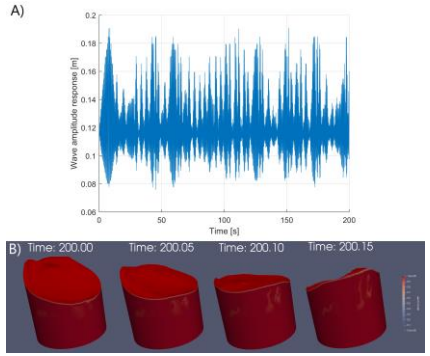
[1] Hopfinger, E. J., & Baumbach, V. (2009). Liquid sloshing in cylindrical fuel tanks. Progress in Propulsion Physics, 1, 279-292. doi: 10.1051/eucass/200901279

SLOSHING DYNAMICS: MODES

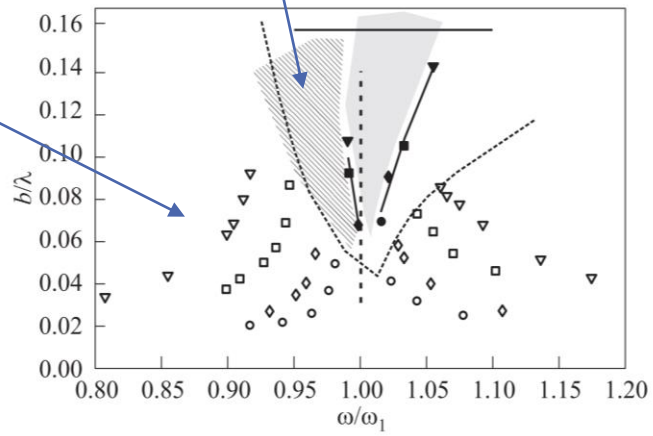
Sloshing behavior depends on **forcing amplitude** and **forcing frequency**



Linear



Chaotic

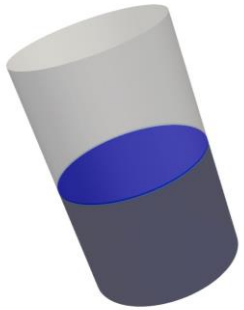


Amplitude-frequency diagram for four different forcing amplitudes [1]

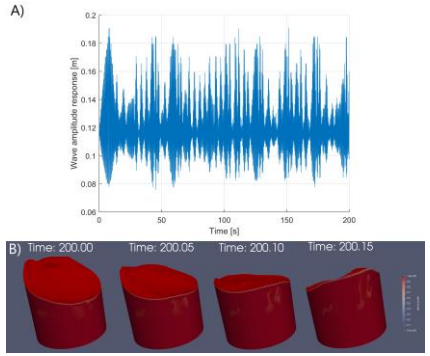
[1] Hopfinger, E. J., & Baumbach, V. (2009). Liquid sloshing in cylindrical fuel tanks. Progress in Propulsion Physics, 1, 279-292. doi: 10.1051/eucass/200901279

SLOSHING DYNAMICS: MODES

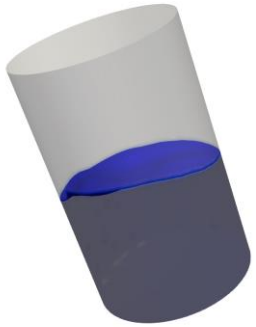
Sloshing behavior depends on **forcing amplitude** and **forcing frequency**



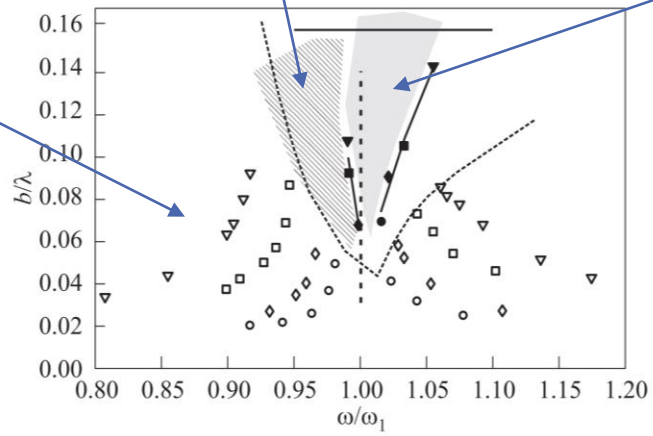
Linear



Chaotic



Swirling



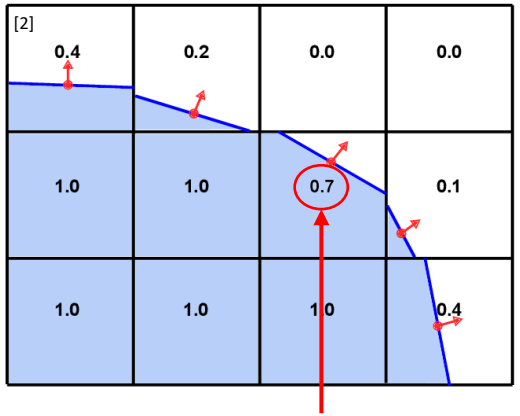
Amplitude-frequency diagram for four different forcing amplitudes [1]

[1] Hopfinger, E. J., & Baumbach, V. (2009). Liquid sloshing in cylindrical fuel tanks. Progress in Propulsion Physics, 1, 279-292. doi: 10.1051/eucass/200901279

SLOSHING DYNAMICS: CFD MODELLING

- Sloshing still follows N-S equations
- Problem with two-phase flows: interface is a mathematical "**discontinuity**"
  - Level-Set (LS)
  - Eulerian-Eulerian
  - Volume of Fluid (VoF) → *interFoam*

```
Description
    Solver for 2 incompressible, isothermal immiscible fluids using a VOF
    (volume of fluid) phase-fraction based interface capturing approach,
    with optional mesh motion and mesh topology changes including adaptive
    re-meshing.
```



Volume fraction: alpha = V_{liq} / V_{cell} →

$$\frac{\partial \alpha}{\partial t} + \frac{\partial (\alpha u_j)}{\partial x_j} = 0$$

+

Numerical method to avoid **numerical smearing** of the interface

[2] Scheufler, H., & Roenby, J. (2021). TwoPhaseFlow: An OpenFOAM based framework for development of two phase flow solvers. arXiv preprint arXiv:2103.00870.

OBJECTIVE OF THE TUTORIAL

- ❑ Understand how to setup a basic two-phase sloshing case
- ❑ Add variable acceleration input to an existing OpenFOAM solver
- ❑ Basic photorealistic post-processing

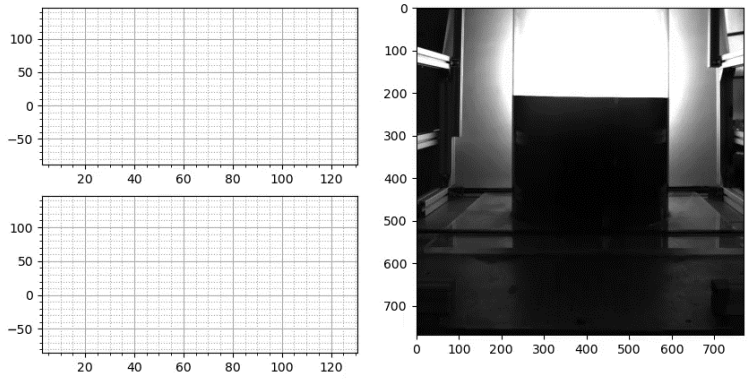
TABLE OF CONTENTS

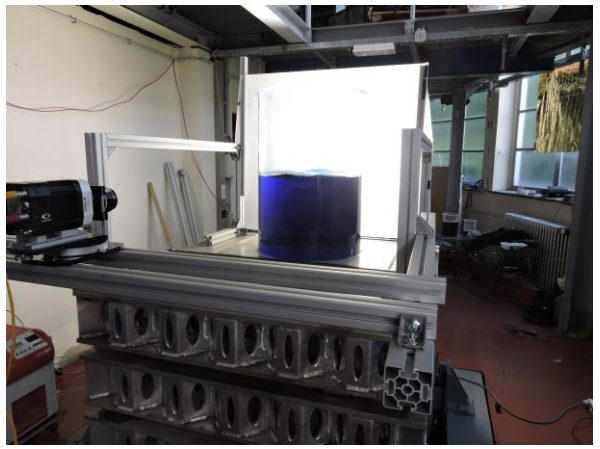
SLOSHING DYNAMICS: REFERENCE TEST CASE

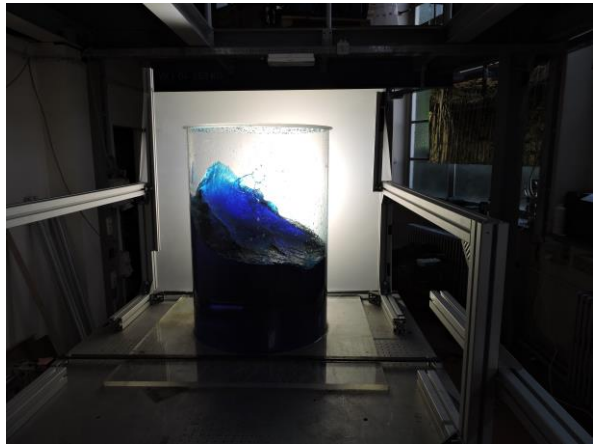
Vertical tank subjected to sinusoidal motion along one axis

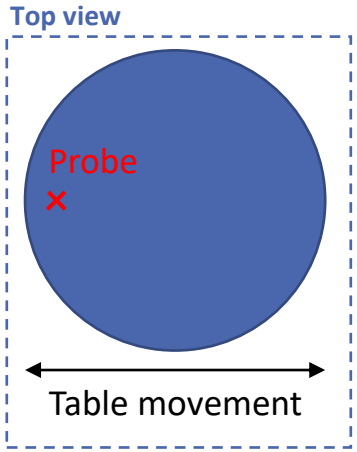
- Experiments performed at **von Karman Institute** SHAKESPEARE facility (**SH**aking **A**paratus for **K**inetic **E**xperiments of **S**loshing Projects with **EA**rthquake **Re**production)

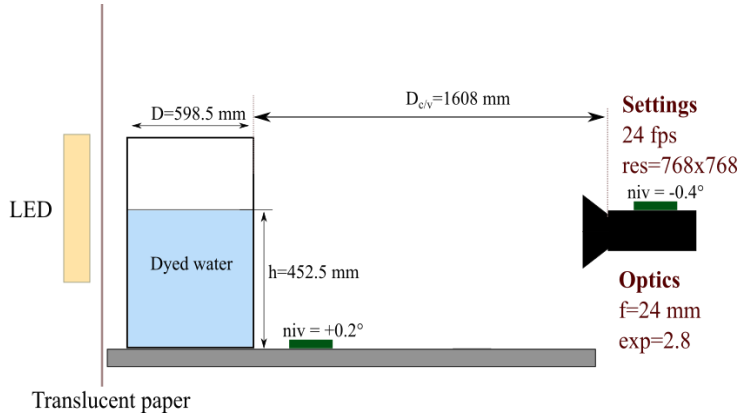
- Available measurements
  1. Interface level
  2. Shaking table movement











Credits: Jean Muller [3]

[3] Muller, J., Planquart, P., & Laboureur, D. (2023). Experimental investigation of the transition between linear to chaotic regimes of sloshing. Multiphase Science and Technology, 35(4).

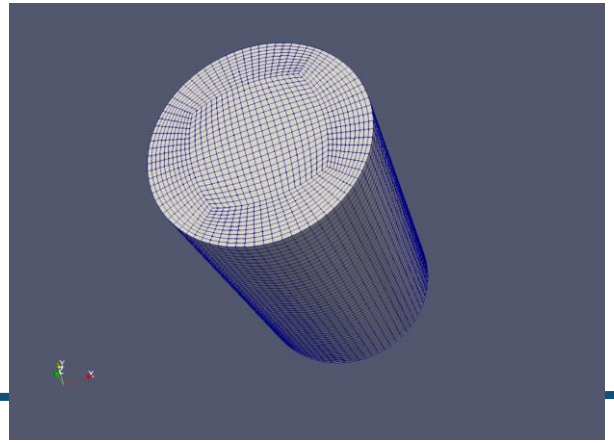
SLOSHING DYNAMICS: CASE SETUP

```
>> blockMesh
```
← Commands to type in your terminal (without >>)

OpenFOAM v9 will be used in this seminar

1. Download material from *git*
   - *To setup a simulation it is common practice to start from a tutorial which has all (or most) of the features we need in the simulation. In this case the tutorial multiphase/interFoam/laminar/damBreak in a good start.*
2. Go to the working directory
3. Build your mesh `>> blockMesh`
   - Use the *blockMeshDict* to customize your mesh

     The dictionary file provided will build a cylindrical structured mesh with the dimensions of the experimental cylindrical tank (D = 0.6 m, h = 0.96 m)
     The domain is axial-symmetric, but the problem isn't (sloshing is along an axis perpendicular to the axis of symmetry)!
     We could reduce the computational cost by cutting the domain in half (not done in this tutorial)
4. Check your mesh `>> paraFoam`



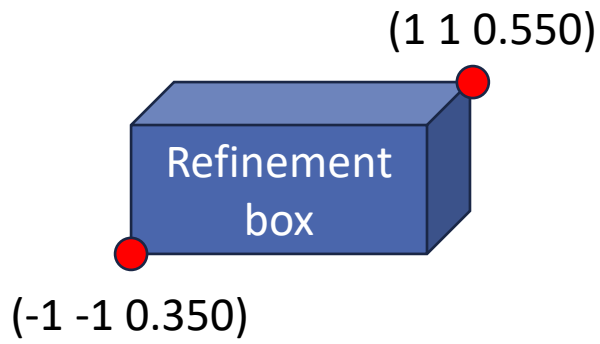
SLOSHING DYNAMICS: CASE SETUP

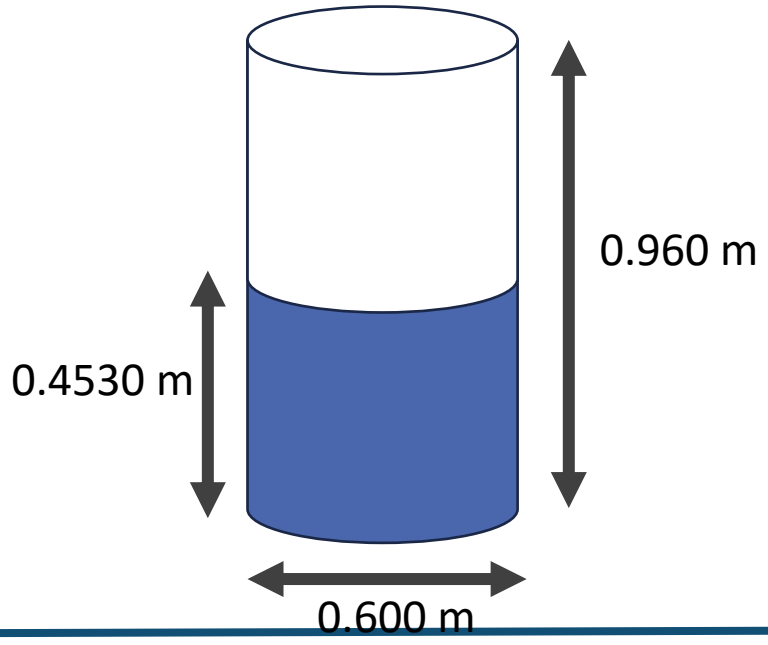
5. Refine the mesh in the vicinity of the interface

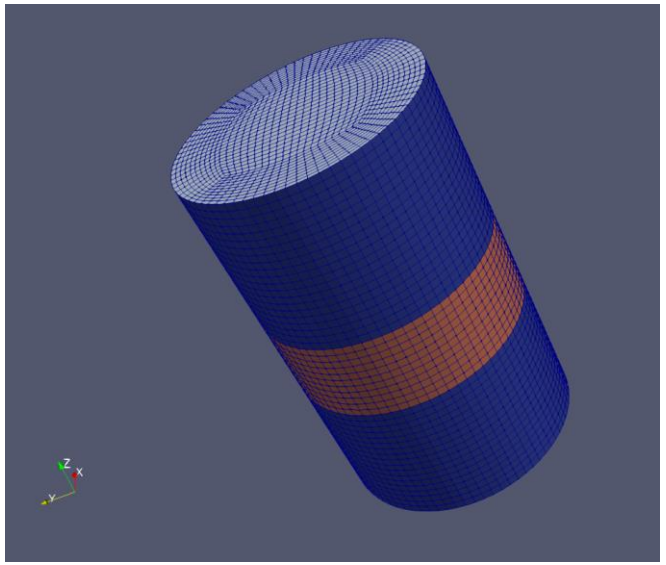
In the problem we want to simulate we want to maximize the accuracy of the interface position, therefore it is desirable to increase the mesh resolution in the cells where we expect the interface to be during its motion
Another option could be to use the *dynamic mesh refinement* (not done in this tutorial)

- topoSet will perform the operations listed in the file *system/topoSetDict*
- In this case we want to select the cells close to the interface (i.e. the cells that lay in the box with extremity points (-1 -1 0.350)(1 1 0.550) and assign them to a cell set (i.e. a group of cells) named *refineSet*
- Set the coordinates of the "box" we want to refine in the *system/topoSetDict* file, change the name of the set then run topoSet

```
>> topoSet
```







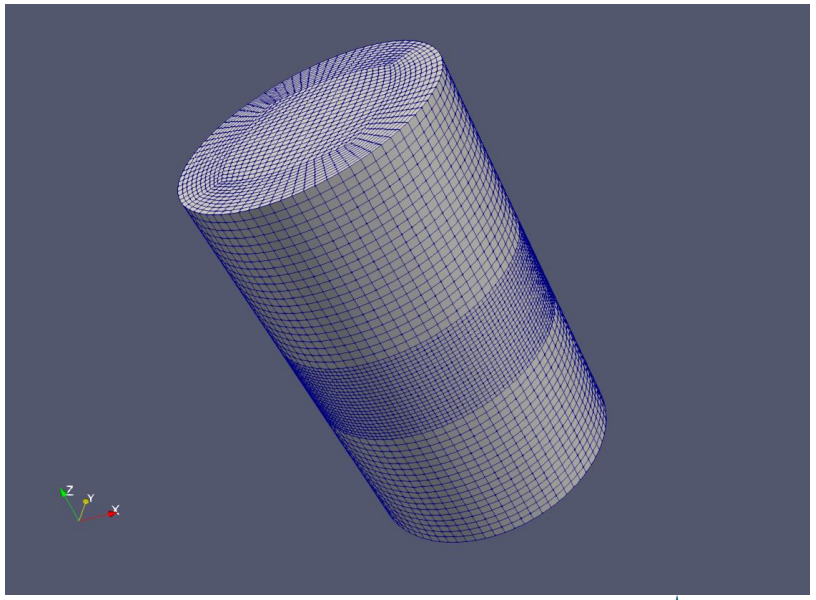
SLOSHING DYNAMICS: CASE SETUP

5. Refine the mesh in the vicinity of the interface

In the problem we want to simulate we want to maximize the accuracy of the interface position, therefore it is desirable to increase the mesh resolution in the cells where we expect the interface to be during its motion
Another option could be to use the *dynamic mesh refinement* (not done in this tutorial)

- *refineMesh* will read the dictionary file *system/refineMeshDict* and perform the mesh refinement operations listed. In this case we want to refine the cells belonging to the cellSet named *refineSet.*
- In addition, we can specify along which direction we want to refine
- Open *system/*refineMeshDIct, change the name of the set to refine, set the proper directions to refine, the run *refineMesh - overwrite*

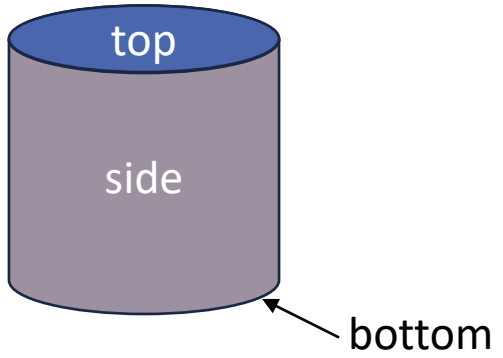
```
>> refineMesh -overWrite
```



6. Check your mesh

```
>> paraFoam
```

SLOSHING DYNAMICS: CASE SETUP



6. Setup boundary conditions

   In the *blockMeshDict* file we have defined 3 named boundary conditions
   - side
   - top
   - bottom

   For each we must specify boundary conditions for the following variables:
   - *alpha.water:* we will assume a 90 degree contact angle (neglecting surface tension effects)
   - *p_rgh*: water-wall at equilibrium
   - *U*: zero velocity

   Open these three files (0/*) then modify the boundary conditions name and type

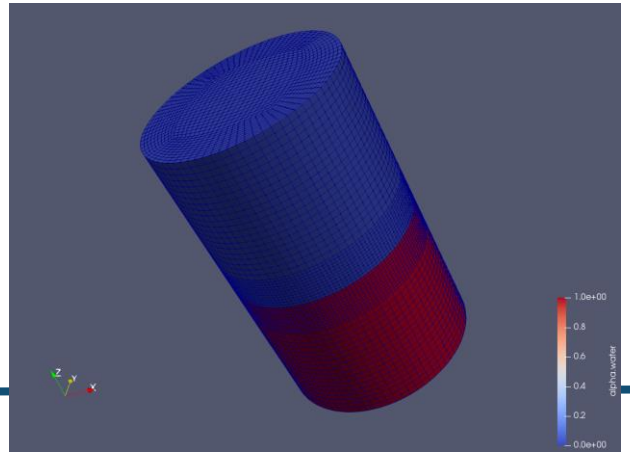
7. Initialize liquid volume fraction internal field

   To modify the internal field of alpha.water, we will use the utility setFields. Set fields reads the file *system/setFieldsDict* and modifies the internal fields of all specified variables. In our case we will use a box to select the cells where we want to set the volume fraction to 1.

   Open *system/setFieldsDict*, properly set the box extreme points (remember, the water level in the experiments is 0.453m from the bottom of the tank), then run *setFields*

```
>> setFields
```

8. Check mesh and internal fields



SLOSHING DYNAMICS: CASE SETUP

9. Setup mesh movement

   The experiment we want to simulate uses a sinusoidal oscillation with

   - Amplitude: 0.00399 m
   - Omega: 7.1484 rad/s

   Mesh movement is controlled by the dynamic mesh libraries. In our case we want to move the whole domain, so the *solidBody* libraries with *oscillatingLinearMotion* function will do the trick!

   Set up the proper parameters in the file *constant/dynamicMesh*

10. Setup gravity acceleration

    Gravity acceleration is read from the file *constant/g* (our domain axis is along z, with positive direction pointing up)

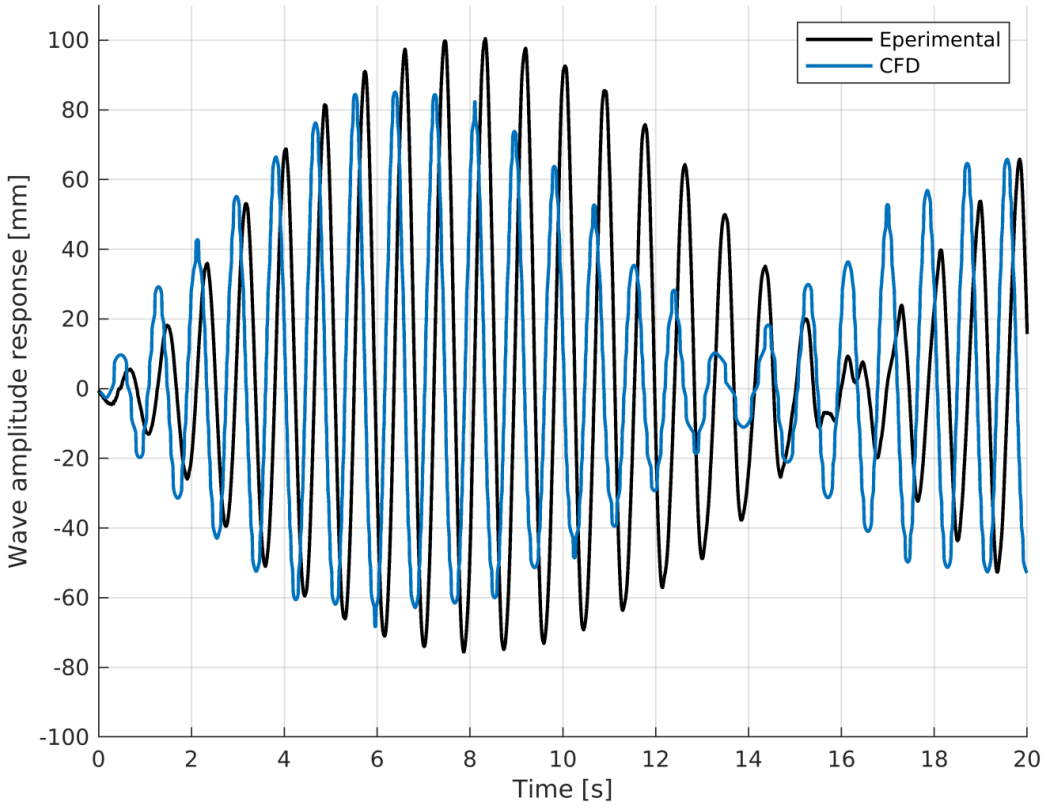
11. Record liquid level on specified location

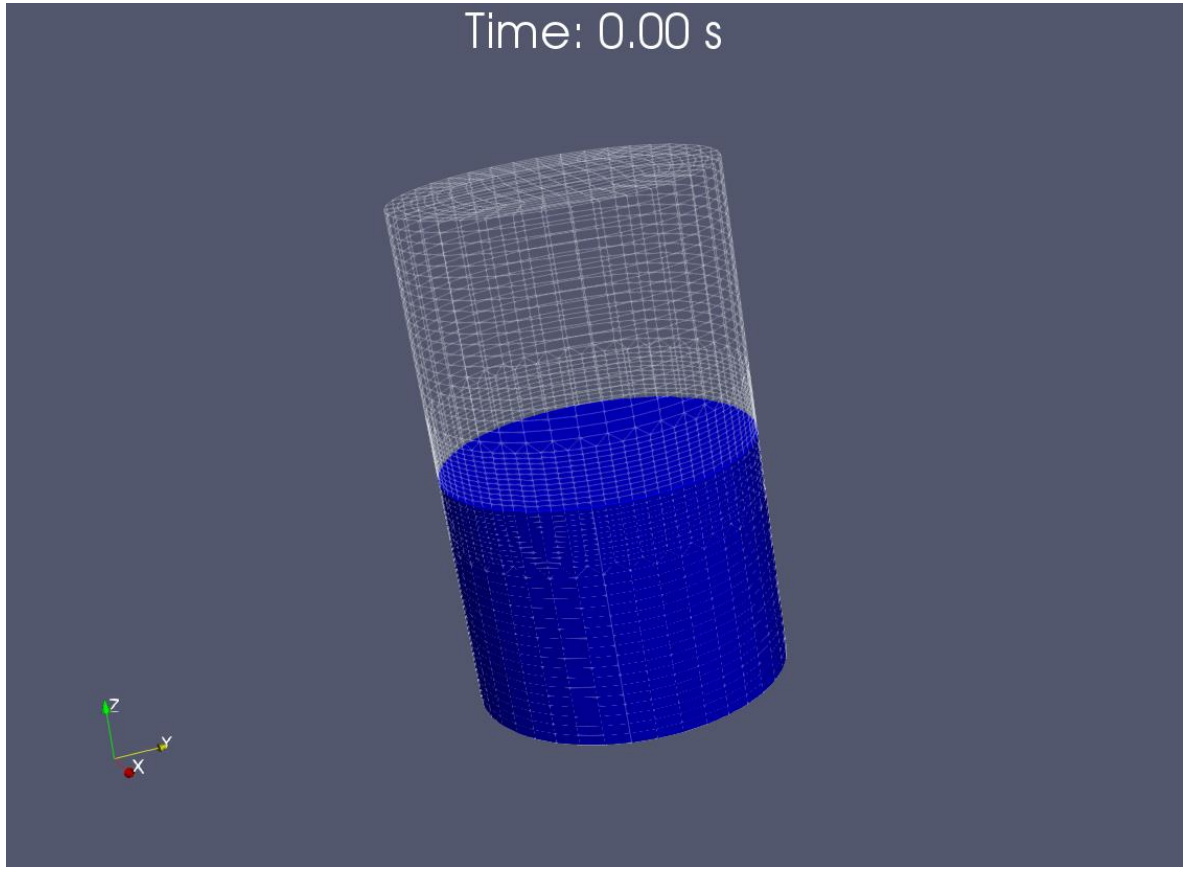
    To compare the results of the simulation with the experimental measurements it is useful to save to a file the position of the free surface in the same location where it was recorded during the experiments. (Already done for you in *system/controDict,* check the file and try to understand!)

Run interFoam!

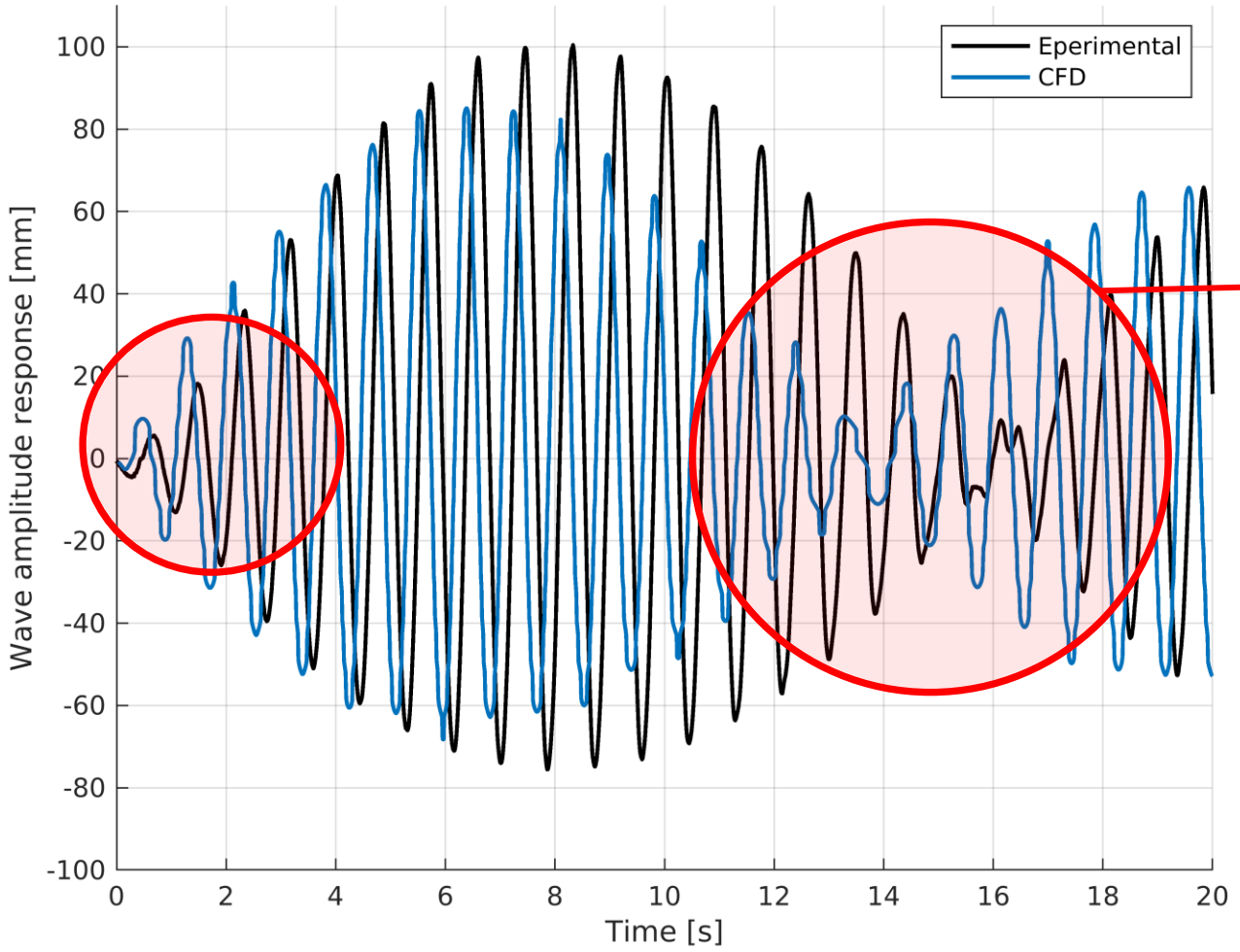
```
>> foamJob interFoam
```

SLOSHING DYNAMICS: RESULTS

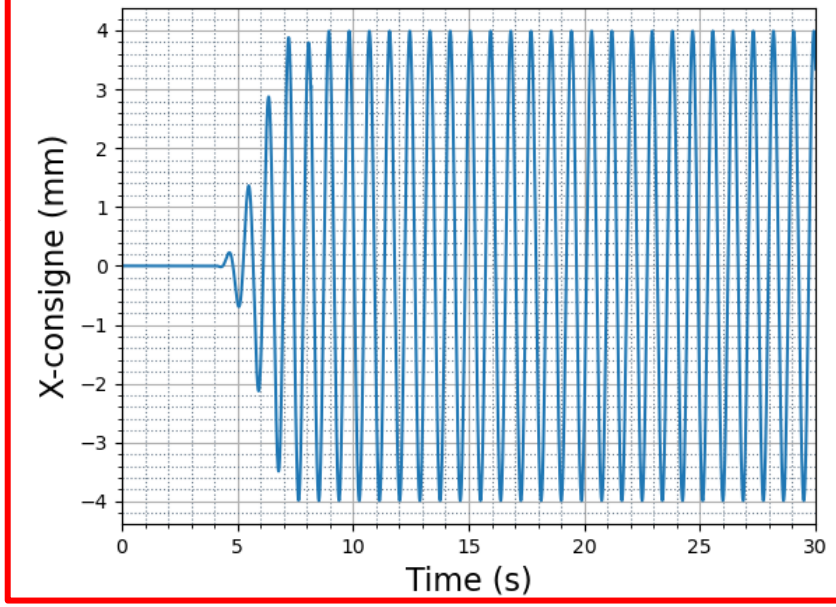




SLOSHING DYNAMICS: RESULTS

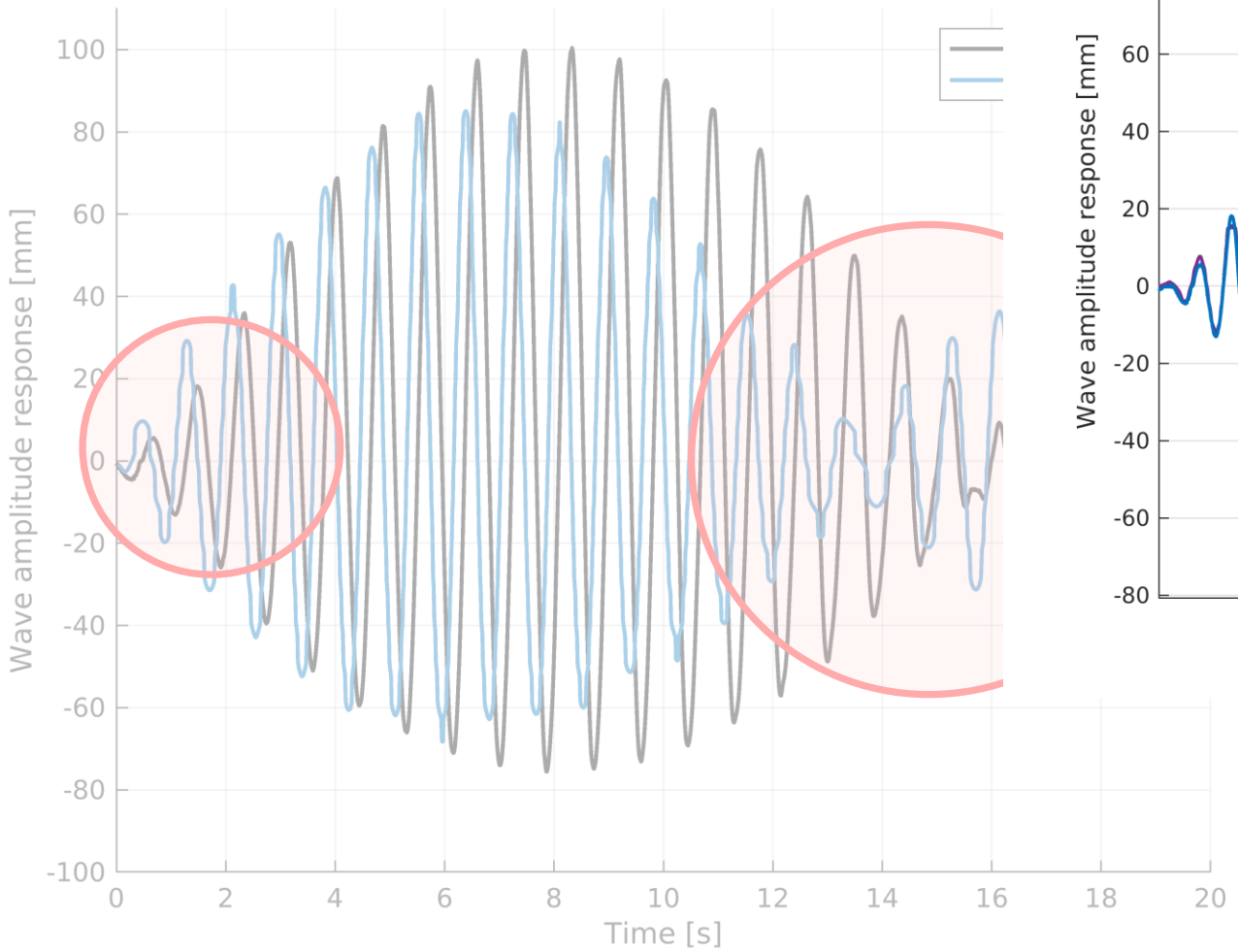


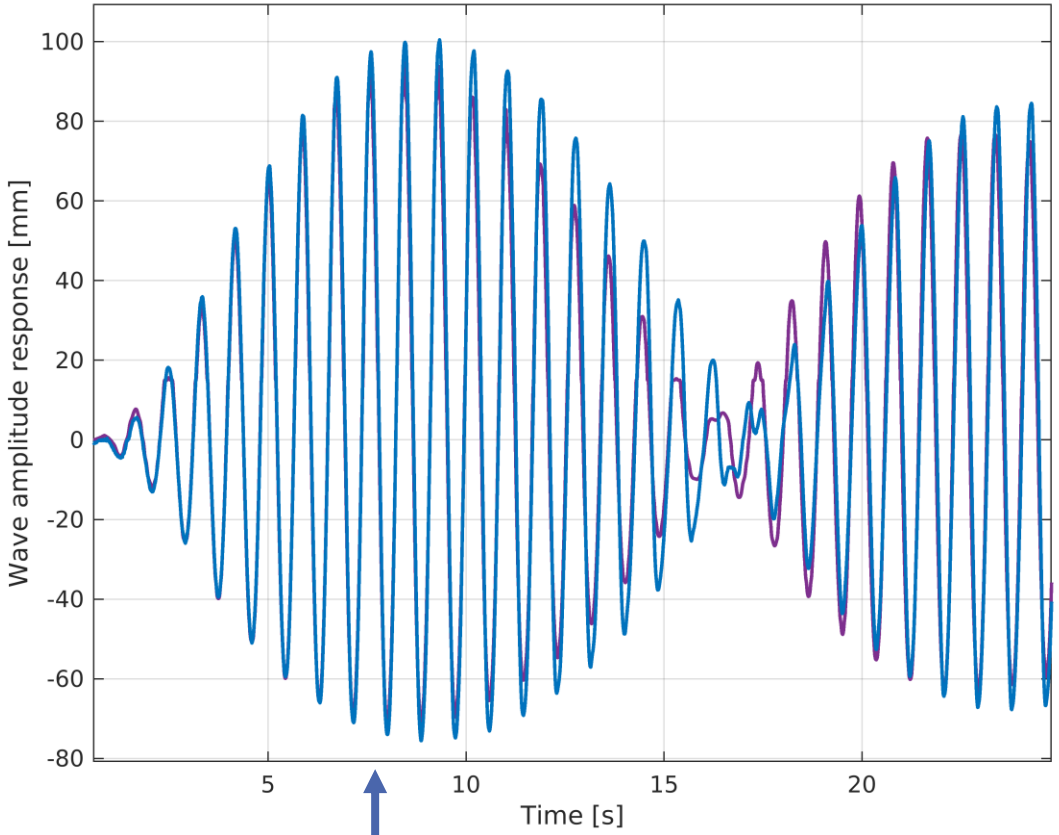
Input on Shakespeare sloshing table 4 periods before max



- **Solution**: input experimental table displacement

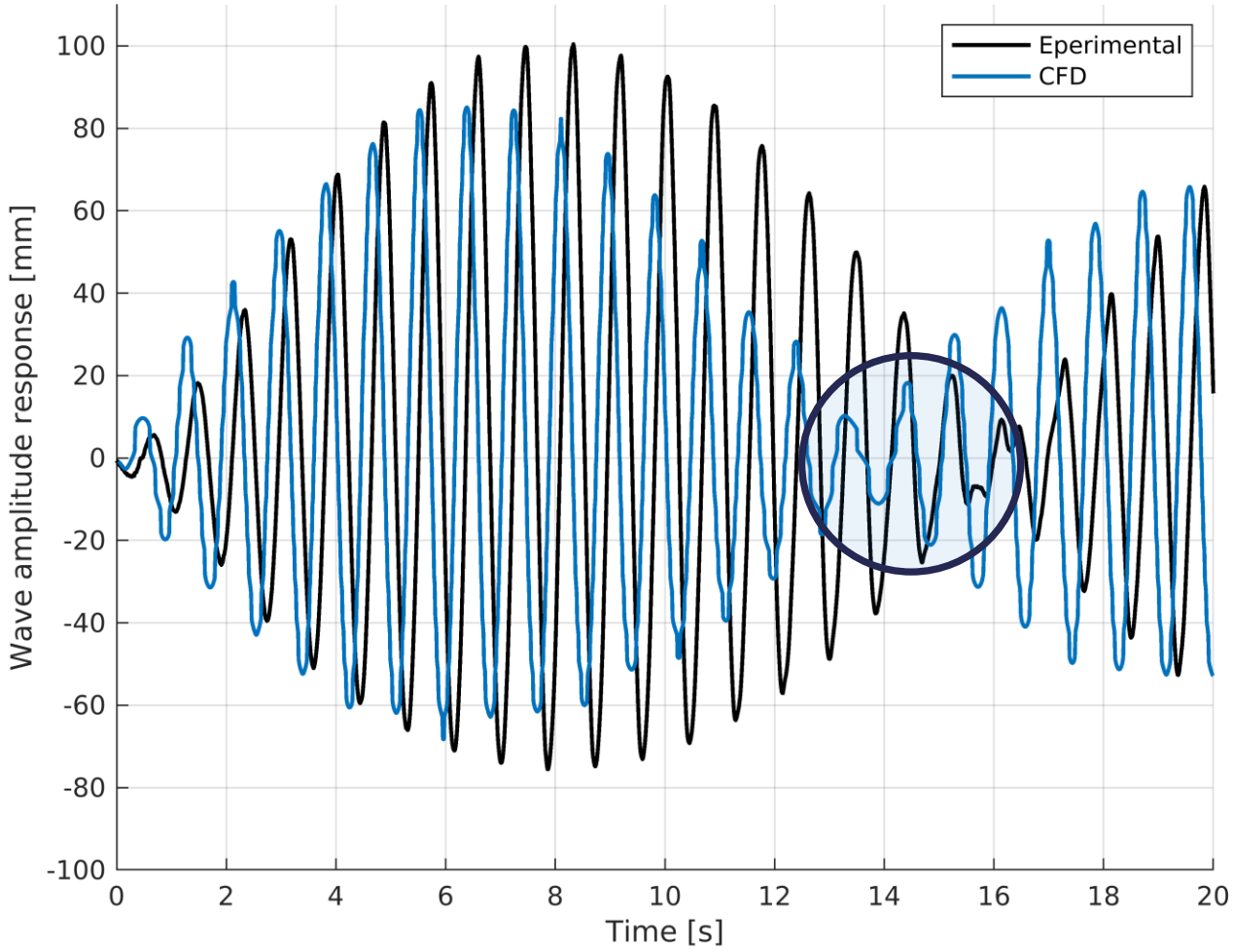
SLOSHING DYNAMICS: RESULTS





- **Solution**: input experimental table displacement

SLOSHING DYNAMICS: RESULTS



Mesh resolution limits interface tracking accuracy

- Increase mesh resolution → increase computation time
- Use dynamic mesh refinement

TABLE OF CONTENTS

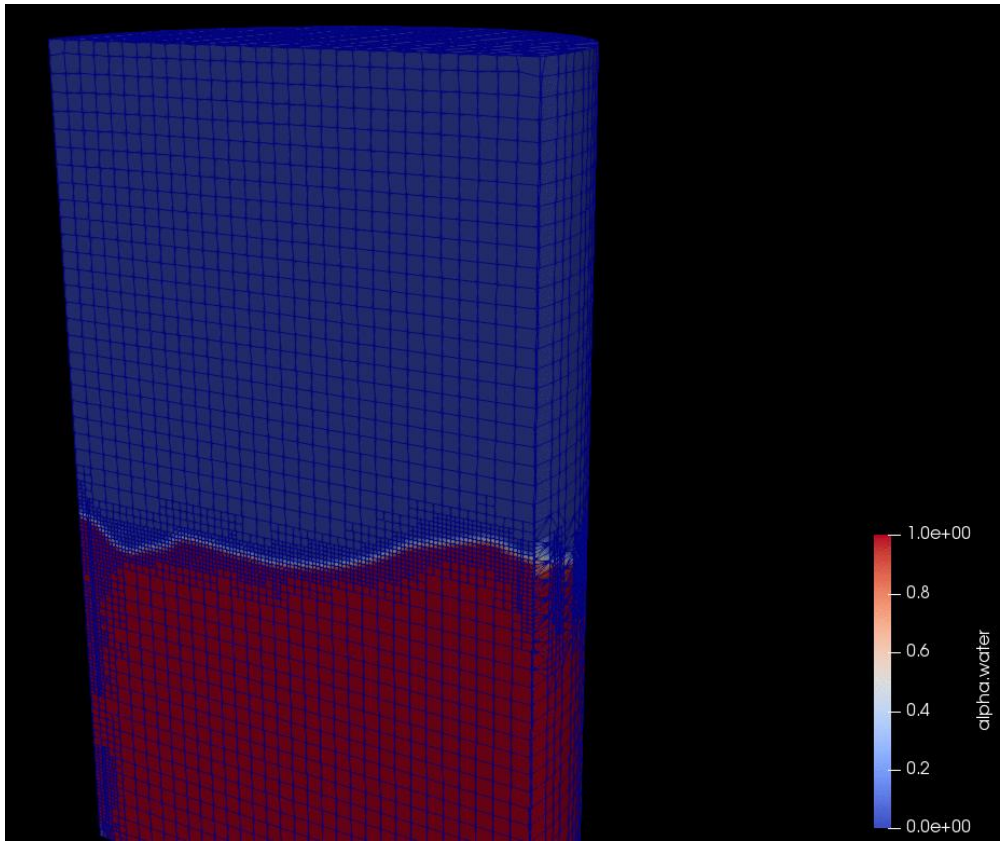
SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- In the current version of OpenFOAM is not possible to use both **mesh movement** and **dynamic mesh refinement** since they belong to the same “family” of libraries “dynamic mesh”.

- Possible strategies:

  - Code a combined **dynamic mesh refinement/solid body motion** library

  - Model the tank motion as **variable acceleration** (what is experienced by the fluid)



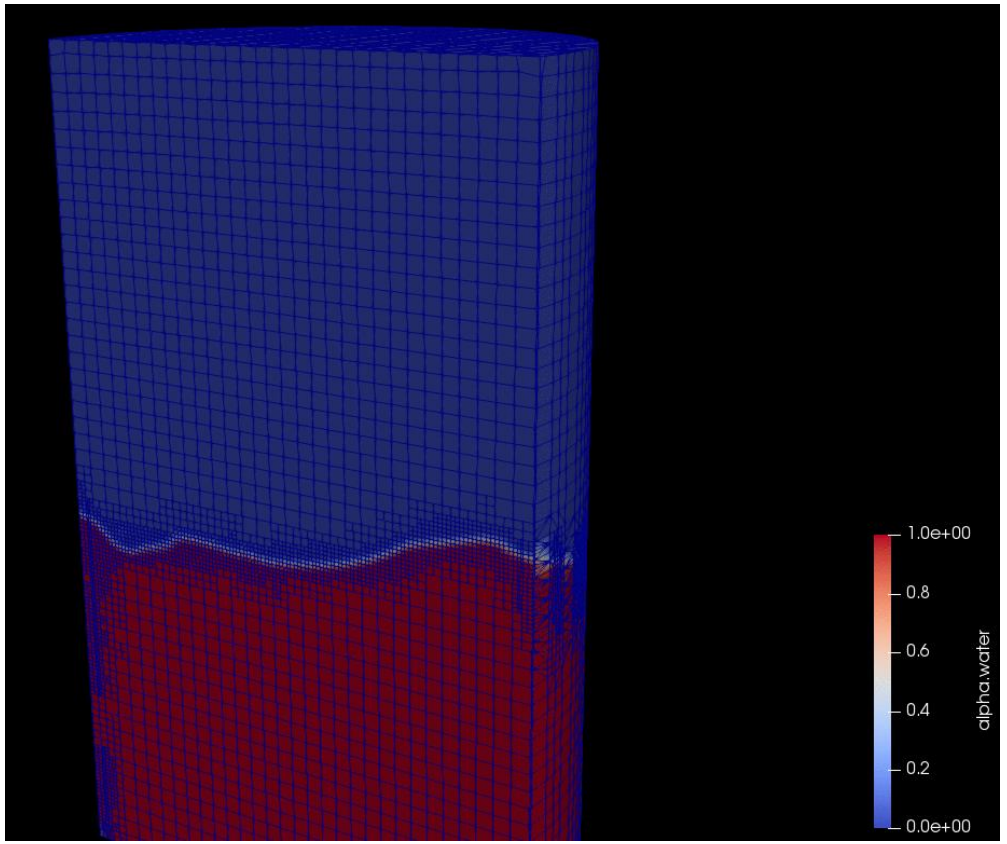
SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

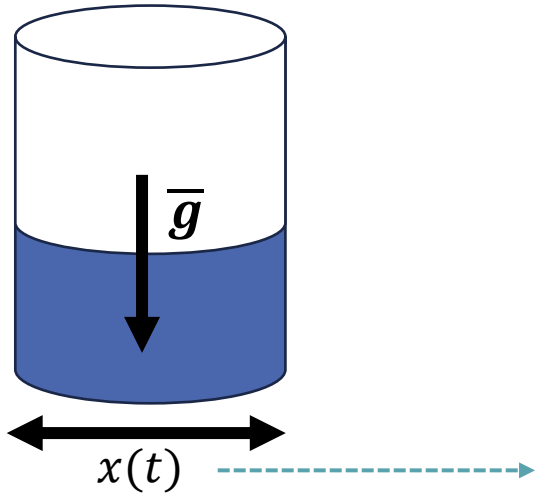
- In the current version of OpenFOAM is not possible to use both **mesh movement** and **dynamic mesh refinement** since they belong to the same "family" of libraries "dynamic mesh".

- Possible strategies:

  - Code a combined **dynamic mesh refinement/solid body motion** library

  - Model the tank motion as **variable acceleration** (what is experienced by the fluid)





$$a(t) = \frac{d^2x(t)}{dt^2}$$

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- Copy the original *interFoam* solver in a new directory (already done in the files provided)
- Change the name of the solver:

1. Change the name of the directory
```
>> mv interFoam interGFoam
```

2. Go in the directory
```
>> cd interGFoam
```

3. Change the name of the main file
```
>> mv interFoam.C interGFoam.C
```

4. Change the name of the files to use in the *Make/files* directory ------▸ **Make/files**

```
interGFoam.C

EXE = $(FOAM_USER_APPBIN)/interGFoam
```

5. Compile the solve (as it is) to be sure that everything is ok
```
>> wmake
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- Now it's time to modify the solver

createFields.H

```
64    // Construct incompressible turbulence model
65    autoPtr<incompressible::momentumTransportModel> turbulence
66    (
67        incompressible::momentumTransportModel::New(U, phi, mixture)
68    );
69
70
71    #include "readGravitationalAcceleration.H"
72    #include "readhRef.H"
73    #include "gh.H"
74
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- Now it's time to modify the solver

createFields.H

```
64    // Construct incompressible turbulence model
65    autoPtr<incompressible::momentumTransportModel> turbulence
66    (
67        incompressible::momentumTransportModel::New(U, phi, mixture)
68    );
69
70
71    #include "readGravitationalAcceleration.H"
72    #include "readhRef.H"
73    #include "gh.H"
74
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- Now it's time to modify the solver

createFields.H

```
64  // Construct incompressible turbulence model
65  autoPtr<incompressible::momentumTransportModel> turbulence
66  (
67      incompressible::momentumTransportModel::New(U, phi, mixture)
68  );
69
70
71  #include "readGravitationalAcceleration.H"
72  #include "readhRef.H"
73  #include "gh.H"
74
```

interGFoam.C

```
54  int main(int argc, char *argv[])
55  {
56      #include "postProcess.H"
57
58      #include "setRootCaseLists.H"
59      #include "createTime.H"
60      #include "createDynamicFvMesh.H"
61      #include "initContinuityErrs.H"
62      #include "createDyMControls.H"
63      #include "createFields.H"
64      #include "createFieldRefs.H"
65      #include "createAlphaFluxes.H"
66      #include "initCorrectPhi.H"
67      #include "createUfIfPresent.H"
68
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- Now it's time to modify the solver

interGFoam.C

```
[…]
int main(int argc, char *argv[])
{
    //The dimensionless scalar gunits is used to add units to acceleration variables
    const dimensionedScalar gunits ("gunits", dimensionSet(0,1,-2,0,0,0,0),1);

    #include "postProcess.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
[…]
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

interGFoam.C

```
[…]

if (!LTS)
    {
        #include "CourantNo.H"
        #include "setInitialDeltaT.H"
    }

    //Reading acceleration dictionary "accelerationDict" present in the constant directory
           IOdictionary accelerationDict
    (
        IOobject
        (
            "accelerationDict",                                    // dictionary name
            runTime.constant(),     // dict is found in "constant"
            mesh,                   // registry for the dict
            IOobject::MUST_READ,    // must exist, otherwise failure
            IOobject::NO_WRITE      // dict is only read by the solver
        )
    );
           word accelerationCoeffs = "accelerationCoeffs";
           word gravityAcceleration = "gravityAcceleration";
           const dictionary& subDict = accelerationDict.subDict(accelerationCoeffs);
           const vector amplitudeAcc = subDict.lookup("amplitude");
           const vector omegaAcc = subDict.lookup("omega");

           Info << "Acceleration amplitude [m] = " << amplitudeAcc << endl;
           Info << "Acceleration omega [rad/s] = " << omegaAcc << endl;

           const vector gravityAcc = accelerationDict.subDict(gravityAcceleration).lookup("gravity");
           Info << "Gravity aceleration [m/s2] = " << gravityAcc << endl;
    //End of acceleration dictionary reading

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;

[…]
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

interGFoam.C

```
[…]
 runTime++;
       Info<< "Time = " << runTime.timeName() << nl << endl;

      //Current acceleration field vector (Harmonic oscillation + constant value) for the 3 components
     g=gunits*amplitudeAcc[0]*Foam::pow(omegaAcc[0],2)*Foam::cos(omegaAcc[0]*runTime.value())*vector(1,0,0)+
     gunits*gravityAcc[0]*vector(1,0,0)+
                       gunits*amplitudeAcc[1]*Foam::pow(omegaAcc[1],2)*Foam::cos(omegaAcc[1]*runTime.value())*vector(0,1,0)+
     gunits*gravityAcc[1]*vector(0,1,0)+
                       gunits*amplitudeAcc[2]*Foam::pow(omegaAcc[2],2)*Foam::cos(omegaAcc[2]*runTime.value())*vector(0,0,1)+
     gunits*gravityAcc[2]*vector(0,0,1);

      //Applying acceleration vector field to the domain
      Info<< "Calculating field g.h\n" << endl;
      volScalarField gh("gh", g & mesh.C());
      surfaceScalarField ghf("ghf", g & mesh.Cf());


      // --- Pressure-velocity PIMPLE corrector loop
      while (pimple.loop())
      {
[…]
```

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- We can now compile the modified solver

```
>> wmake
```

- And then run the same case, but remember to add the *accelerationDict* file in the *constant* directory and the proper settings in the *dynamicMeshDict* file (now we want to refine the mesh)

accelerationDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  9.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      accelerationDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


accelerationCoeffs
{
                amplitude       (0 0.004 0);
                omega           (0 7.1484 0);
}

gravityAcceleration
{
                gravity         (0 0 -9.81);

}
```

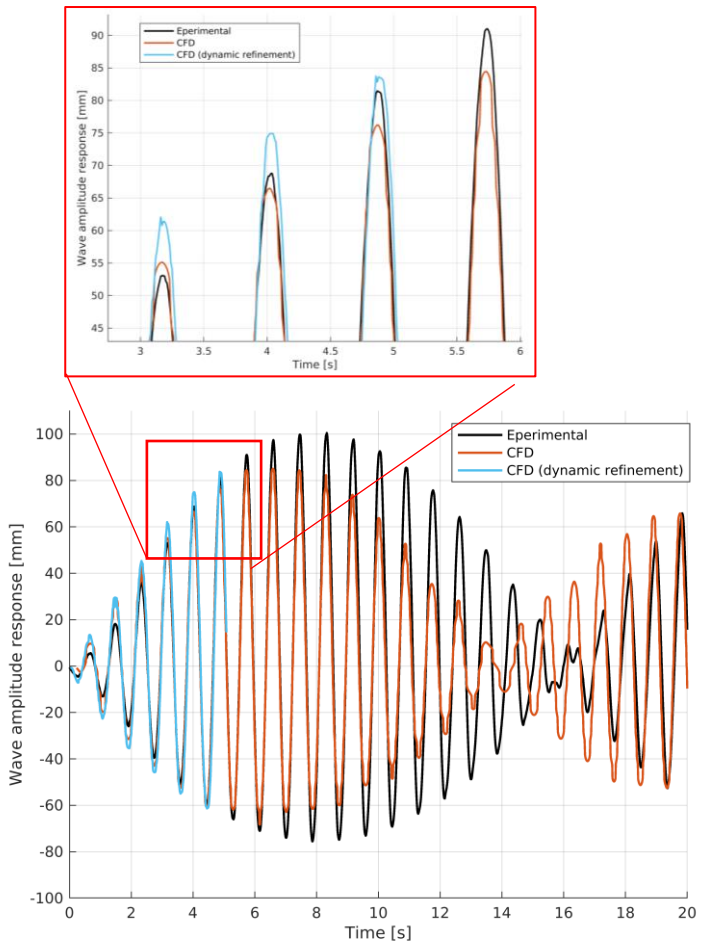
dynamicMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  9
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   dynamicRefineFvMesh;

refineInterval  1; // How often to refine

field           alpha.water; // Field to be refinement on

lowerRefineLevel 0.001; // Refine field in between lower..upper
upperRefineLevel 0.999;

unrefineLevel   10; // If value < unrefineLevel unrefine

nBufferLayers   1; // Have slower than 2:1 refinement

maxRefinement   2; // Refine cells only up to maxRefinement levels

maxCells        200000; // Stop refinement if maxCells reached

correctFluxes
(
    (phi none)
    (nHatf none)
    (rhoPhi none)
    (alphaPhi0.water none)
    (ghf none)
);

dumpLevel       true; // Write the refinement level as a volScalarField

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

SLOSHING DYNAMICS: VARIABLE ACCELERATION SOLVER

- We can now run the new case, then compare the results with the previous run

```
>> foamJob interGFoam
```



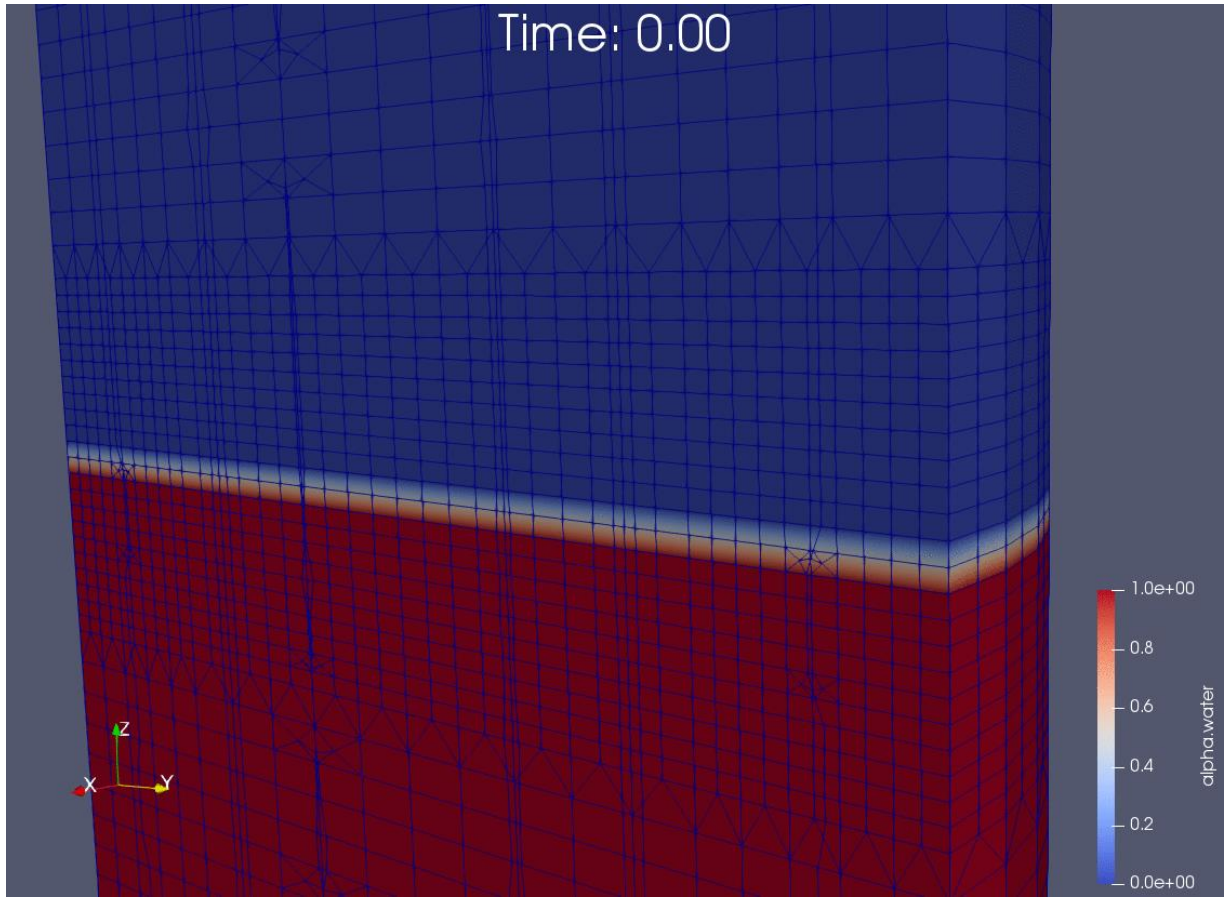
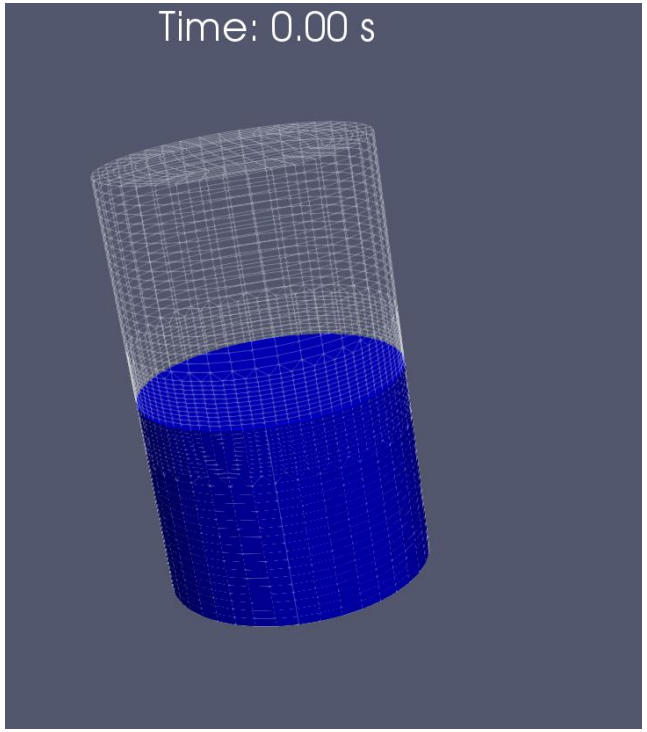


TABLE OF CONTENTS

BASIC PHOTOREALISTIC RENDERING

- Photorealistic rendering provides **clear** and **realistic** visuals, which facilitate effective **communication**

Better for science



More effective for communication

BASIC PHOTOREALISTIC RENDERING

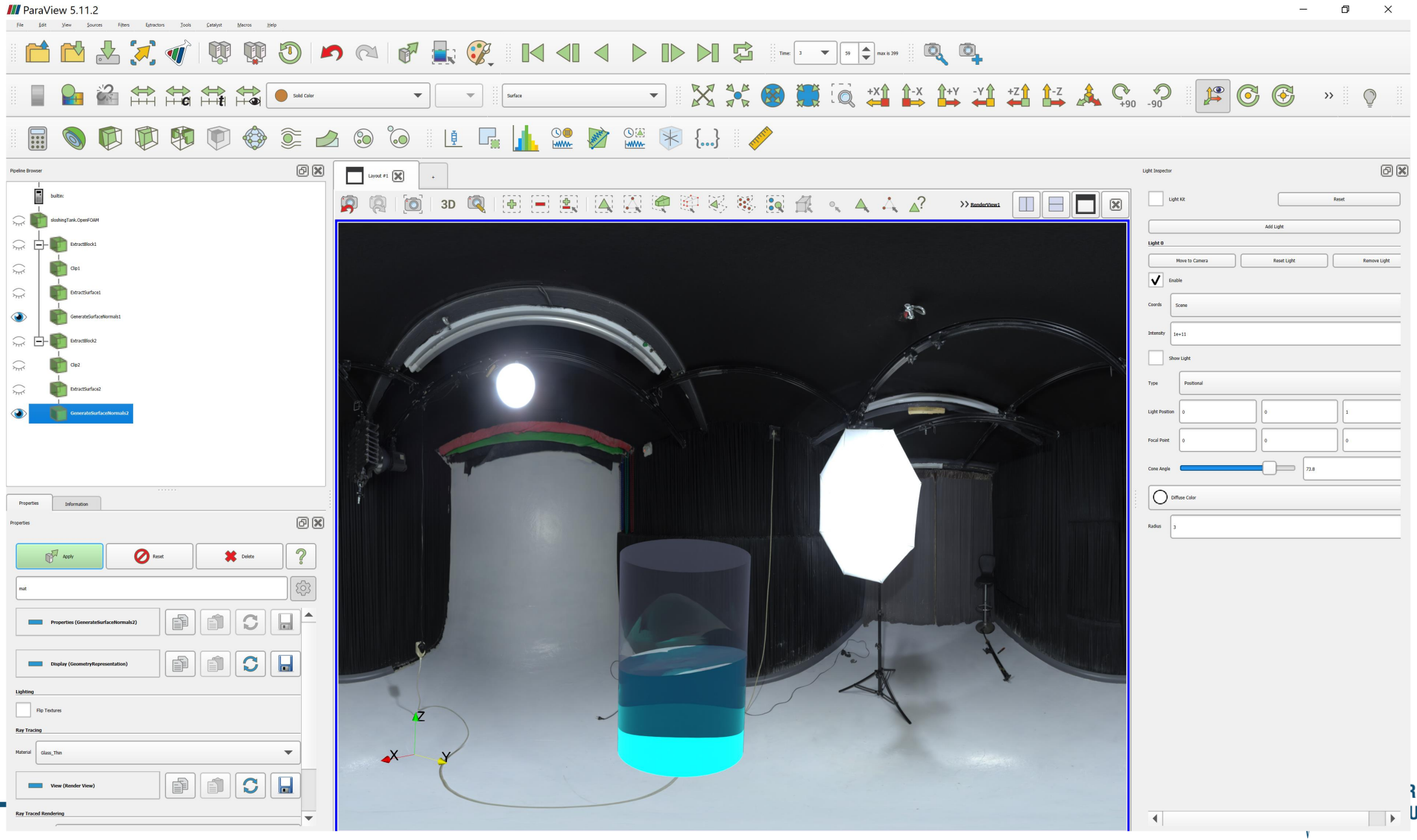
- Photorealistic rendering provides **clear** and **realistic** visuals, which facilitate effective **communication**
- Photorealistic rendering is **NOT CGI**
  - CGI uses deep modelling to have results that “look” physical, but are not physical
  - A **photorealistic rendering** of a CFD simulation “looks” and “is” physical

CGI (Blender) [4]

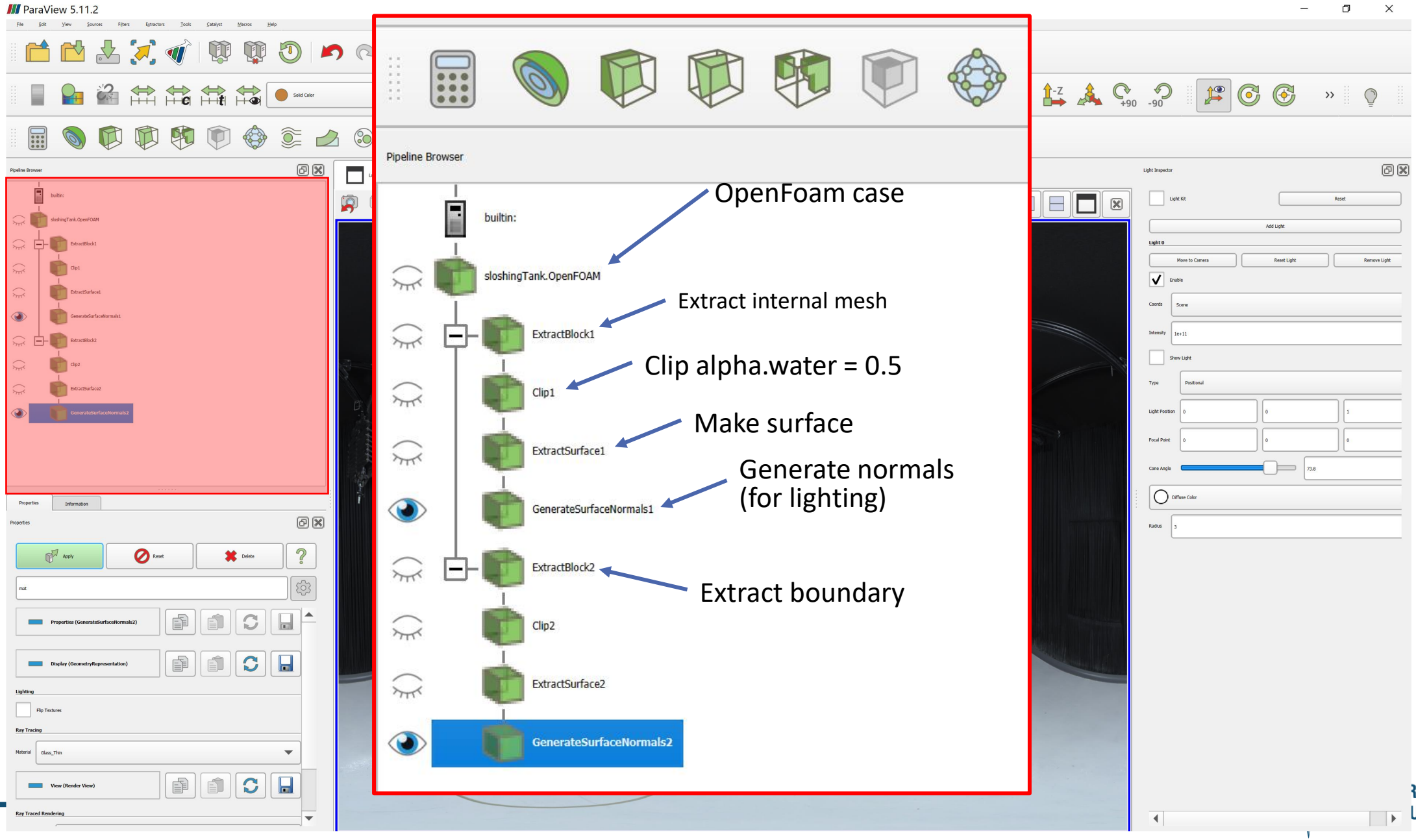
Photorealistic rendering

[4] https://www.youtube.com/watch?v=pukCAUZu15M

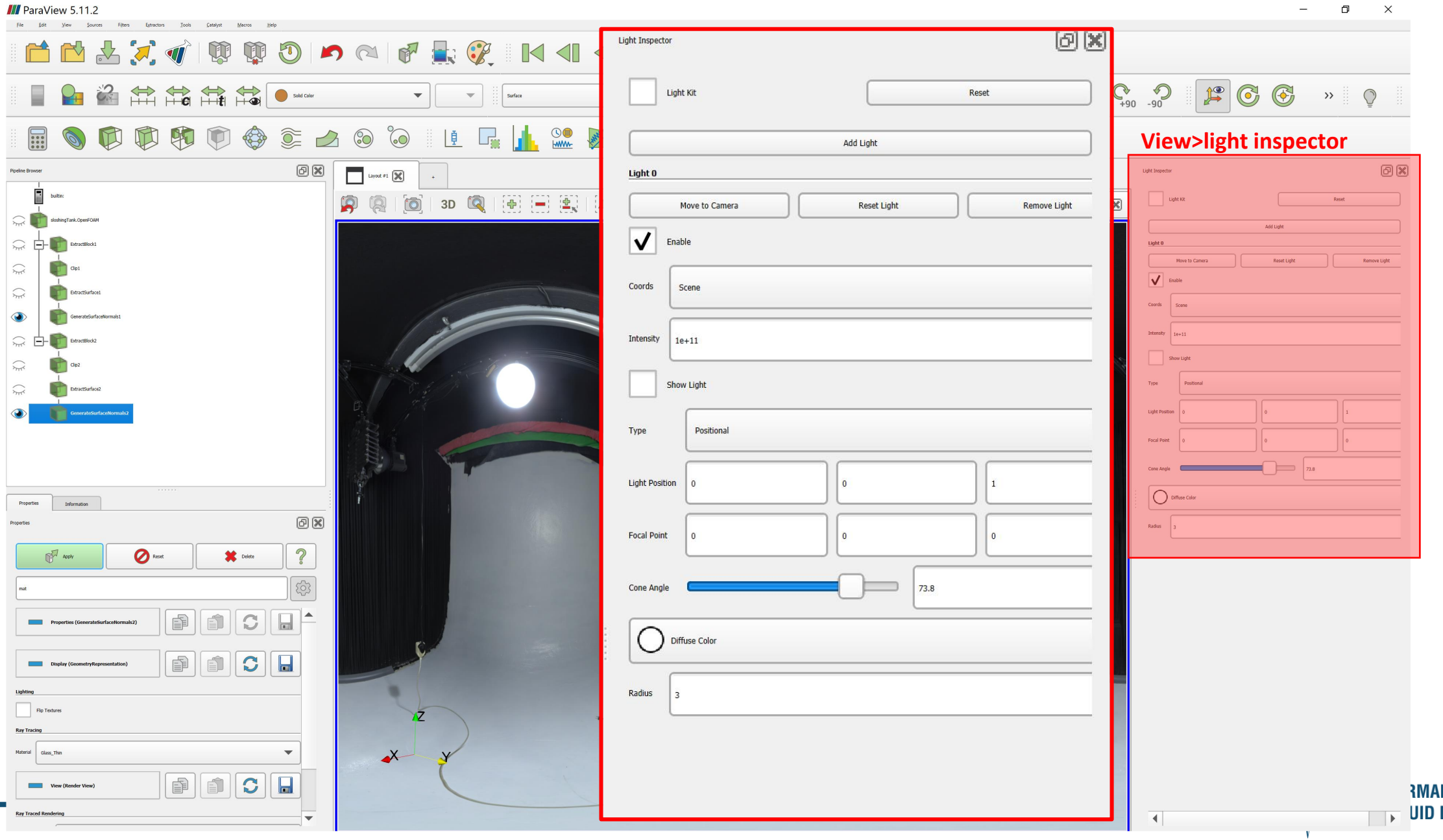
BASIC PHOTOREALISTIC RENDERING



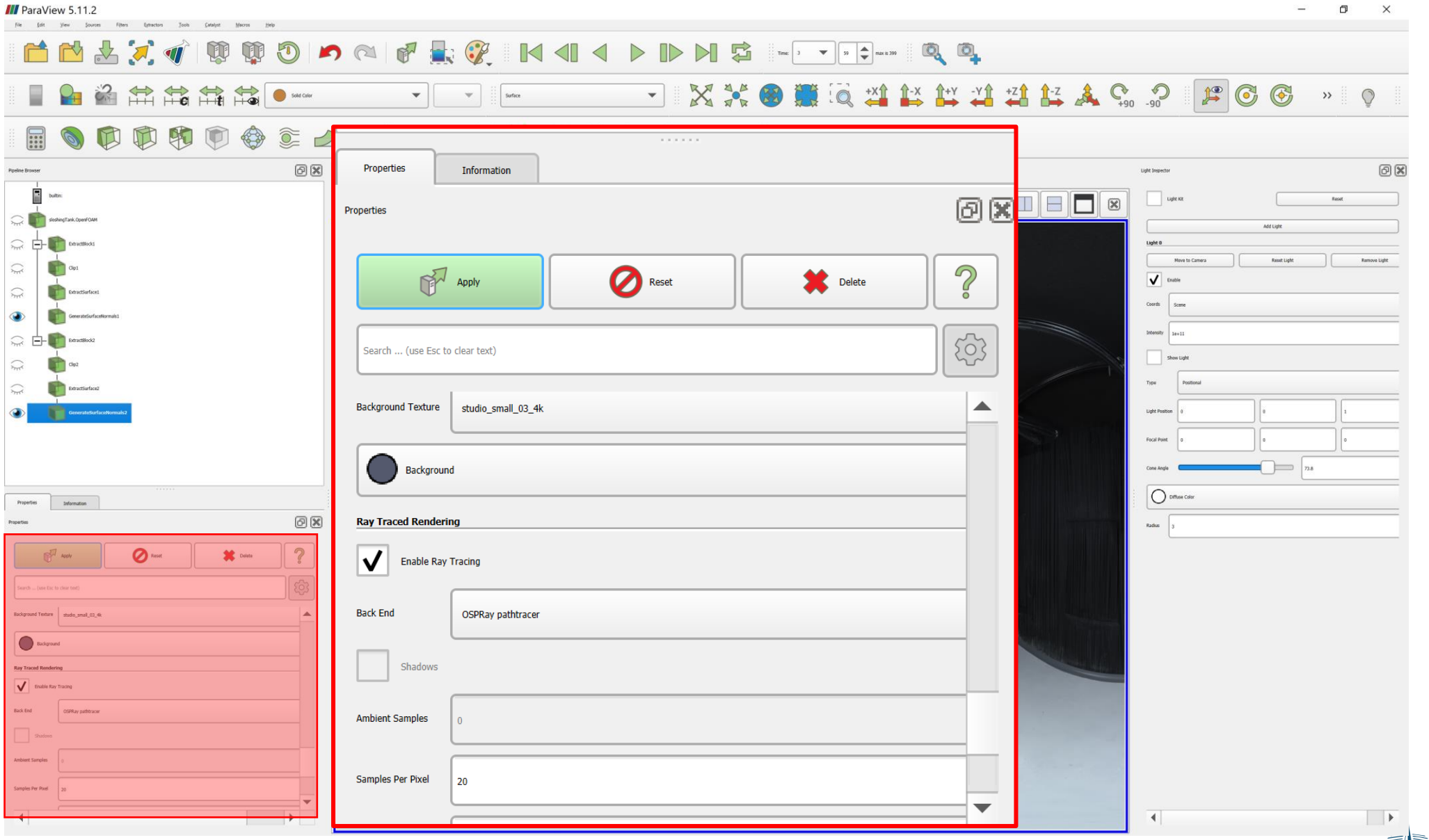
BASIC PHOTOREALISTIC RENDERING



BASIC PHOTOREALISTIC RENDERING



BASIC PHOTOREALISTIC RENDERING



BASIC PHOTOREALISTIC RENDERING

THANK YOU!

SIMULATING SLOSHING DYNAMICS: A PRACTICAL GUIDE USING OPENFOAM

Antonio Cantiani

18th March 2024