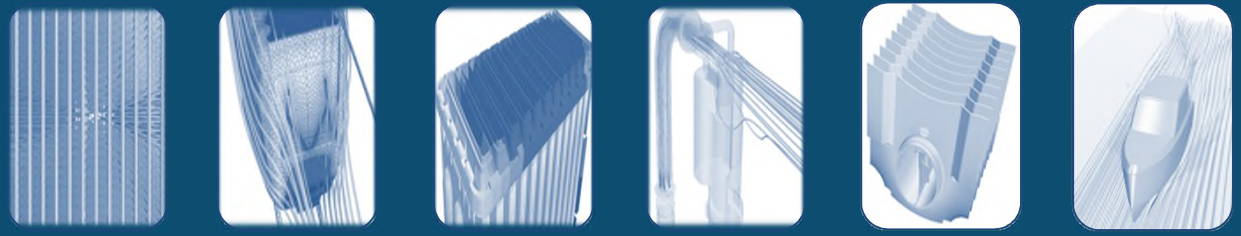
OpenFOAM®
in wastewater applications:
2 - Getting Started

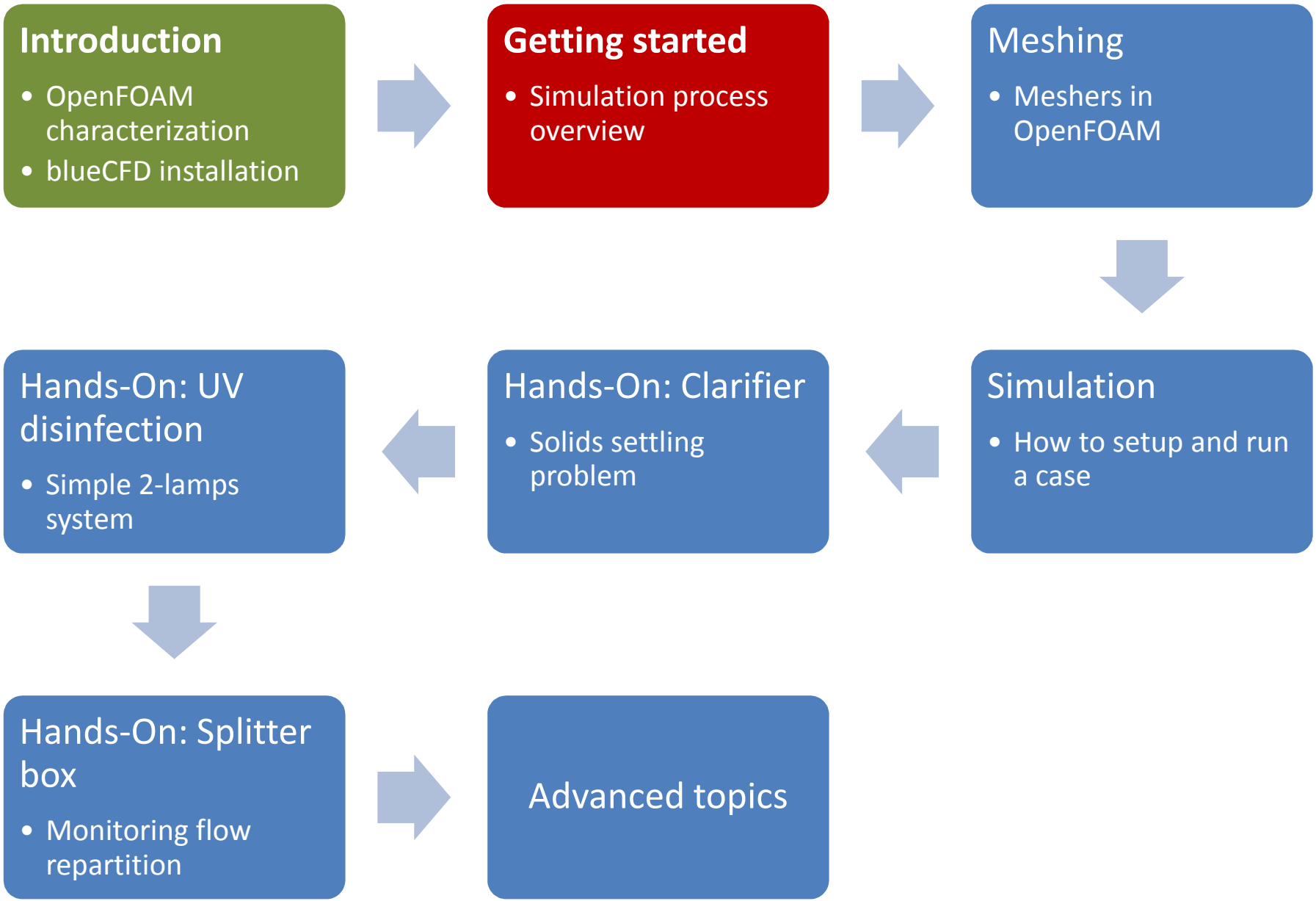
nelson.marques@bluecape.com.pt

13-14th June 2015



bluecape
Computer Applications
in Science & Engineering

Progress


Introduction
• OpenFOAM characterization
• blueCFD installation
Getting started
• Simulation process overview
Meshing
• Meshers in OpenFOAM
Simulation
• How to setup and run a case
Hands-On: Clarifier
• Solids settling problem
Hands-On: UV disinfection
• Simple 2-lamps system
Hands-On: Splitter box
• Monitoring flow repartition
Advanced topics

Contents

1. OpenFOAM Structure
2. Example Case Overview
3. Simulation Case Structure
4. Mesh Generation
5. Preprocessing
    - Model Properties
    - Boundary conditions
    - fvSolution and fvSchemes
6. Simulation
7. Post-processing

OpenFOAM Structure (1/3)

Important Environment Variables in OpenFOAM

$WM_PROJECT_DIR	**path to the OpenFOAM installation**
$WM_PROJECT_USER_DIR	user directory
$FOAM_TUTORIALS	tutorials
$FOAM_SRC	source code directory of OpenFOAM libraries
$FOAM_APP	source code directory of OpenFOAM applications
$FOAM_APPBIN	directory with the compiled OpenFOAM applications
$FOAM_USER_APPBIN	directory with the OpenFOAM applications created by the user
$FOAM_LIBBIN	directory with the compiled OpenFOAM libraries
$FOAM_USER_LIBBIN	directory with the OpenFOAM libraries created by the user
$FOAM_RUN	directory where the user can put his/her cases
echo *variable*	will show you the contents of environment variable Example: echo $WM_PROJECT_DIR

OpenFOAM Structure (2/3)

Important Shell-Aliases in OpenFOAM

foam	**cd $WM_PROJECT_DIR**
foamApps or **app**	cd $FOAM_APP
foamSol or **sol**	cd $FOAM_SOLVERS
foamTuts or **tut**	cd $FOAM_TUTORIALS
foamUtils or **util**	cd $FOAM_UTILITIES
foamsrc	cd $FOAM_SRC/$WM_PROJECT
foam3rdParty	cd $WM_THIRD_PARTY_DIR
foamfv	cd $FOAM_SRC/finiteVolume
lib	cd $FOAM_LIBBIN
run	cd $FOAM_RUN
src	cd $FOAM_SRC
wmSET	. $WM_PROJECT_DIR/etc/bashrc
wmUNSET	. $WM_PROJECT_DIR/etc/config/unset.sh

OpenFOAM Structure (3/3)

OpenFOAM's main folder structure:

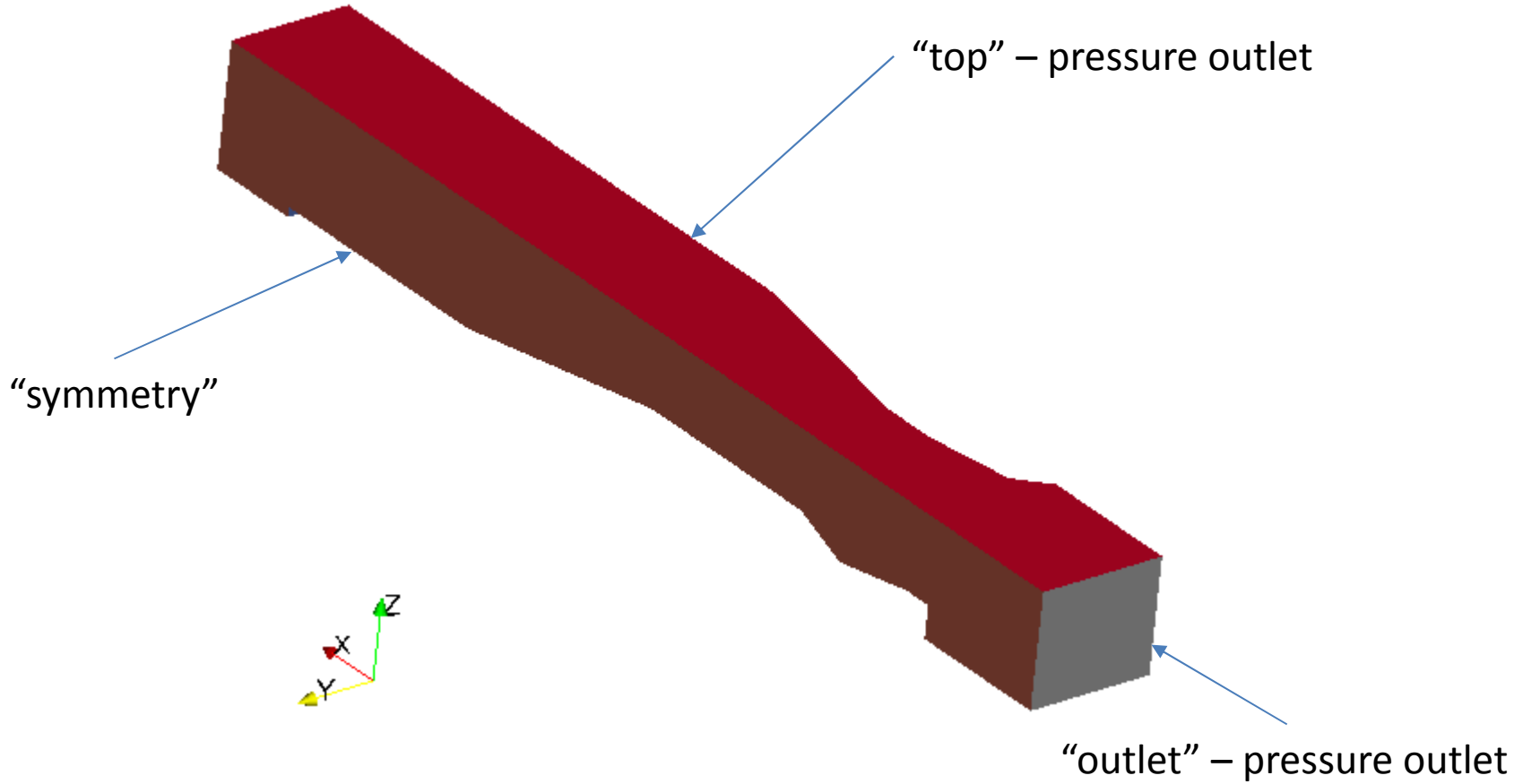
- **applications** – source code for the applications
  - **solvers** – for the solvers
  - **test** – for testing core functions
  - **utilities** –for the utilities (i.e. everything else)
- **bin** – auxiliary scripts for using OpenFOAM
- **doc** –where the documentation is located
- **etc** – scripts for support files (shell environment, etc…)
- **platforms** – where the built binaries are placed
- **src** – the source code of the libraries
- **tutorials** – the tutorial cases
- **wmake** – script infrastructure for building OpenFOAM

Example Case Overview (1/4)

- Case name: halfParshall
- Boundary conditions:
  - Inlet: 375 kg/s
  - Bottom floor and side wall: no-slip
  - Outlet surfaces: pressure outlet
  - Symmetry plane surface: symmetry
- Fluid properties:
  - Water:
    - Density: 999 kg/m^3
    - Dynamic Viscosity: 1.15E-3 Pa.s
  - Air:
    - Density: 1.18 kg/m^3
    - Dynamic Viscosity: 1.855E-5 Pa.s

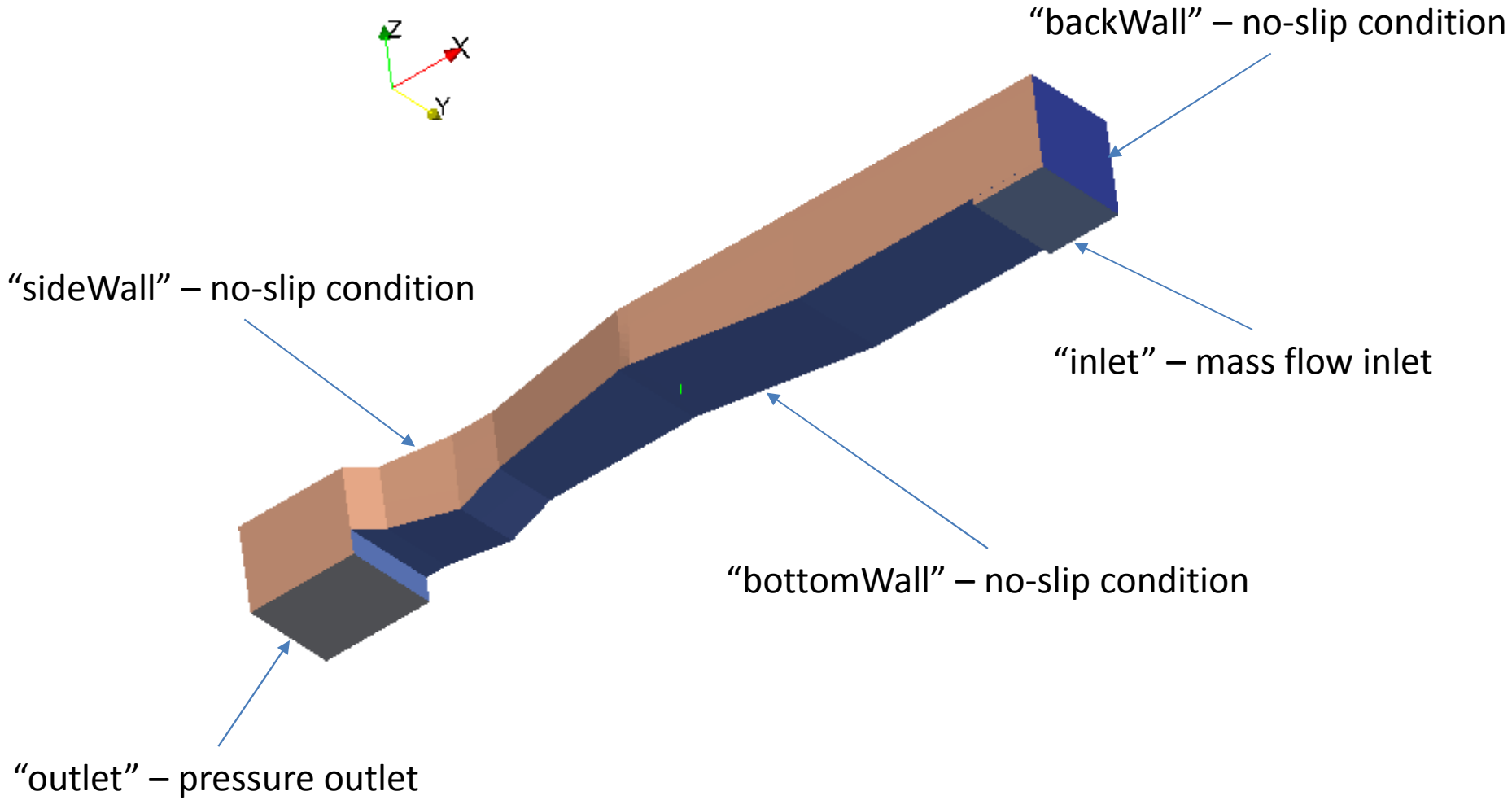
Example Case Overview (2/4)

- Solver type: VOF (volume of fluid)
- Time domain: transient
- Geometry (1/2):



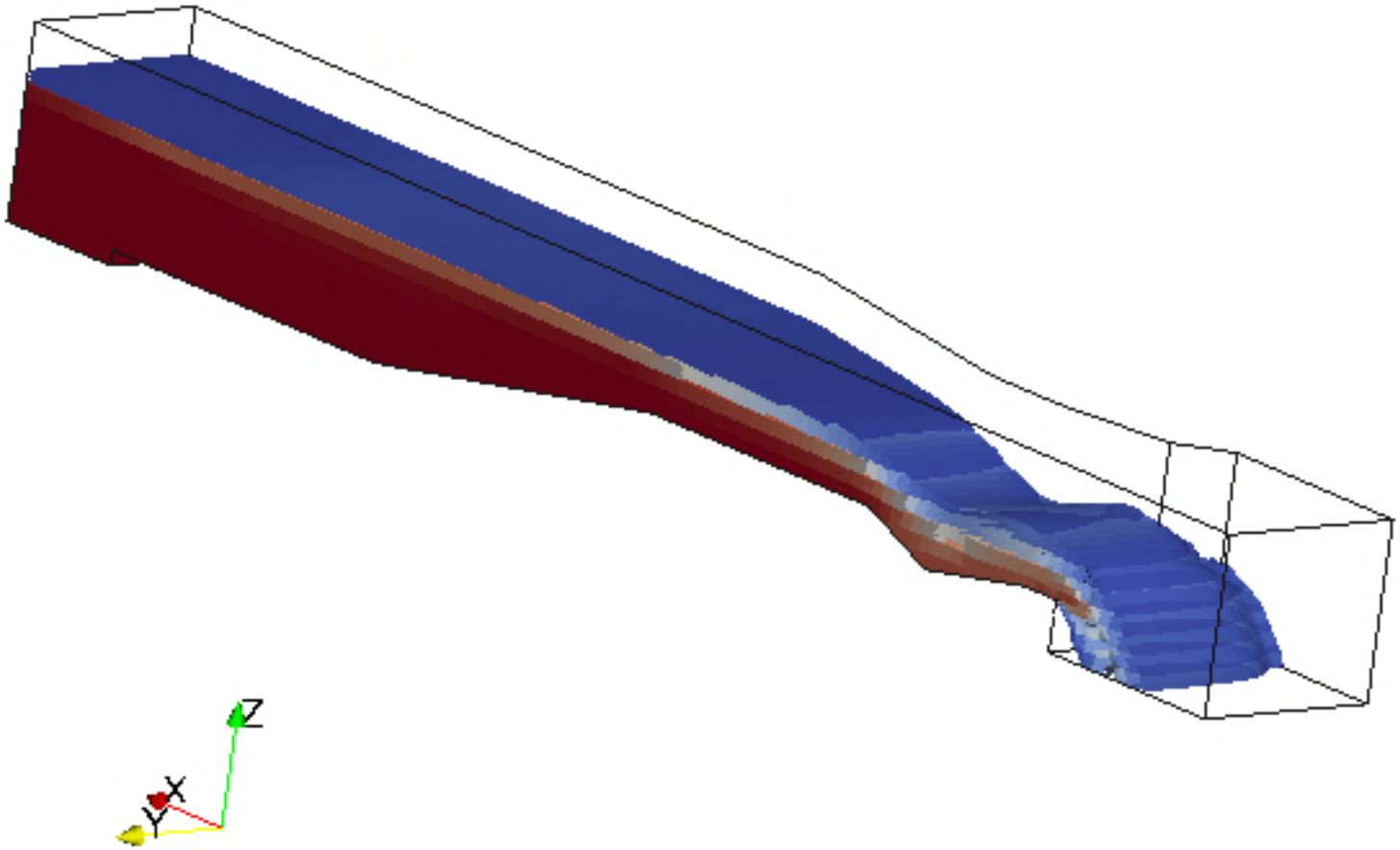
Example Case Overview (3/4)

- Geometry (2/2):



Example Case Overview (4/4)

- Objective:



Simulation Case Structure (1/6)

The case definition is now clear. Now what?

- We select the solver which suits the case characterization. On this case `interFoam`.
- Search a tutorial case that uses that same solver, either from the default tutorial collection ($FOAM_TUT) or through a web search, and copy it across.
- Start adjusting settings (boundary conditions, initialization and numerical parameters) to suit our needs.

Naturally, for meshing to be done, the CAD has to be available as well.

Simulation Case Structure (2/6)

Folder structure in OpenFOAM for our case:

- **halfParshall**
  - **0.org** – original definitions of the initial time step
    - **U** – velocity field
    - **p** – pressure field
    - *etc…*
  - **constant** – e.g. physical properties
    - **polymesh** – polyhedral mesh files
    - **triSurface** – geometrical models
  - **system** – numerics and run-time control
  - *time directories* – examples: 0, 0.1, 1, 2, 3 and so on.

Simulation Case Structure (3/6)

halfParshall/0.org:

- **U** – velocity field
- **p_rgh** – pressure field
- **alpha.water** – phase fraction field
  - 1 = 100% water
  - 0 = 100% not water (air in our case)
- **epsilon** – turbulent dissipation rate field
- **k** – turbulent kinetic energy field
- **nut** – turbulent dynamic viscosity field

Requirement before running the solver:

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

Simulation Case Structure (4/6)

halfParshall/constant:

- **g** – gravity configuration (acceleration vector)
- **RASProperties** – RANS turbulence model configuration
- **transportProperties** – physical properties of the fluids
- **turbulenceProperties** – turbulence category: RAS or LES
- **polyMesh**
  - **blockMeshDict** – dictionary file for **blockMesh**
  - all other files are respective to the mesh, i.e. automatically created, including:
    - **boundary** – geometrical boundary conditions
- **triSurface**
  - **halfParshall.org.stl** – original geometrical model, in STL format

Simulation Case Structure (5/6)

halfParshall/system – essential configuration files:

- **controlDict** – runtime controls (start/stop time, etc...)
- **fvSchemes** – finite volume discretization schemes
- **fvSolution** – linear equation solvers and algorithms

Application-specific files:

- **changeDictionaryDict** – to manipulate dictionary files
- **createPatchDict** – to create/remove/manipulate patches
- **decomposeParDict** – for decomposing into subdomains
- **extrudeMeshDict** – for extruding the mesh
- **setFieldsDict** – for manipulating the fields
- **snappyHexMeshDict** – dictionary for **snappyHexMesh**
- **surfaceFeatureExtractDict** – for calculating feature edges

Simulation Case Structure (6/6)

halfParshall – files in the case's root folder:

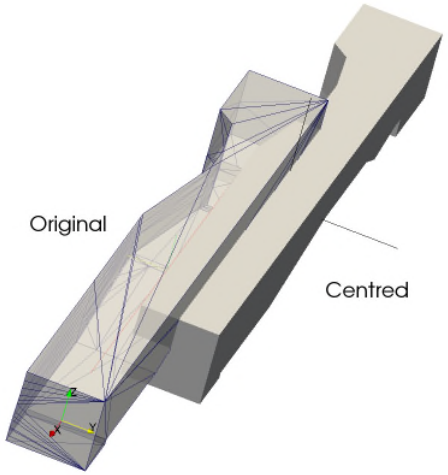
- Scripts for setting up and running the case:
  - **Allrun** – will run all steps, also calls **Allrun.pre**
  - **Allrun.pre** – will preprocess and generate the mesh
- Scripts for resetting the case to the original state:
  - **Allclean** – will reset (clean up) the whole case
  - **Allclean.fields** – will only remove the time snapshots

These scripts are manually created, nonetheless several examples for these files are available in OpenFOAM's "tutorials" folder.

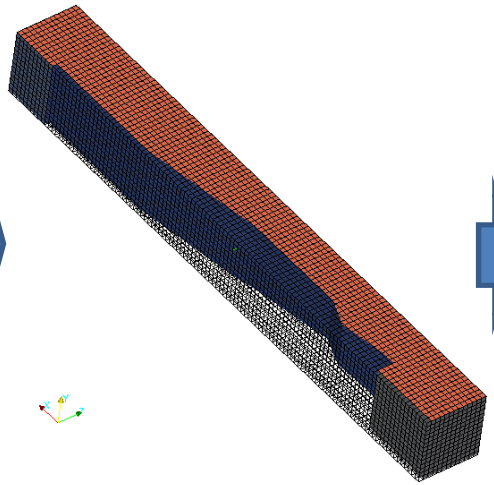
Mesh Generation (1/16)

Meshing steps overview – Visual illustration

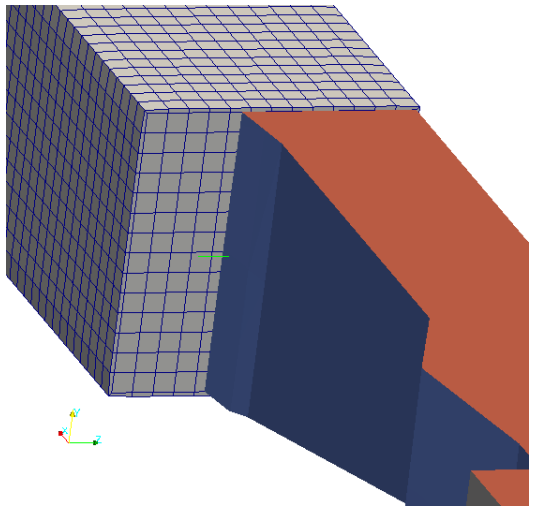
Translate STL



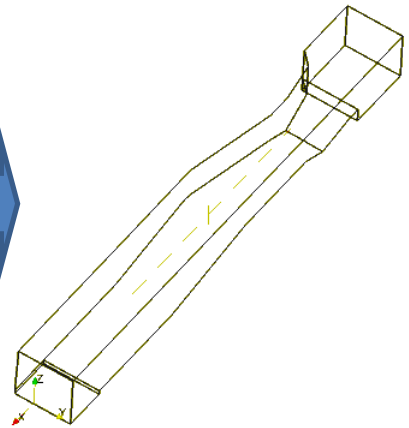
Background Mesh



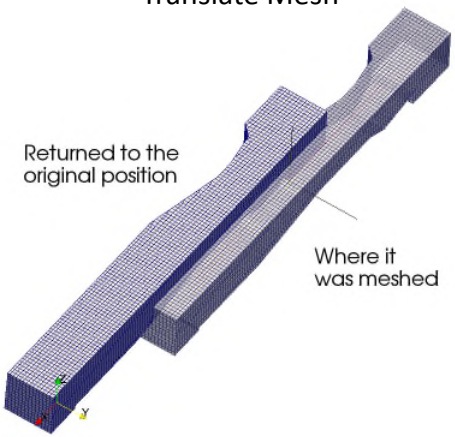
Extrude Mesh



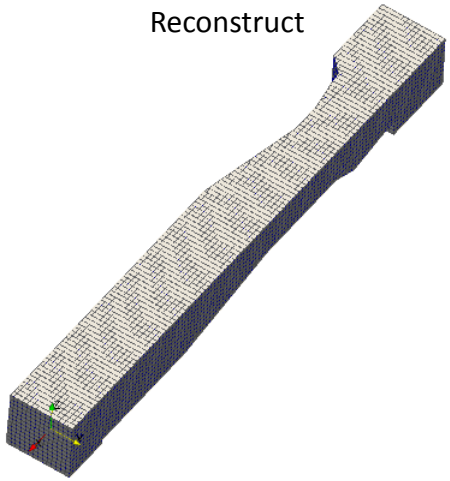
Extract Feature Edges



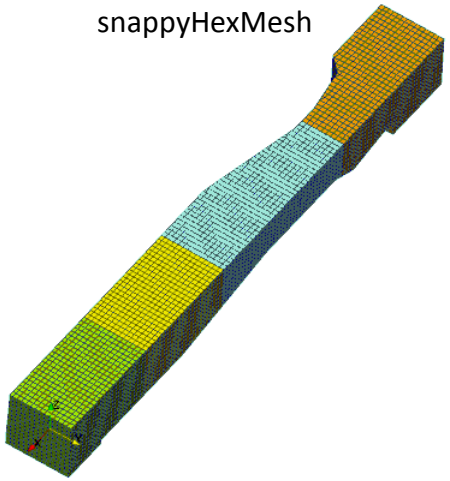
Translate Mesh



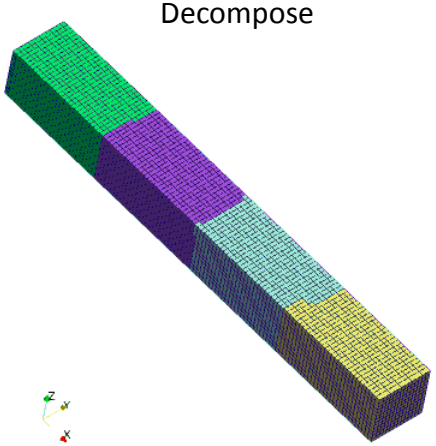
Reconstruct



snappyHexMesh



Decompose



Mesh Generation (2/16)

Meshing steps overview – Contents of the **Allrun.pre** script:

```
runApplication surfaceTransformPoints -translate
'(-4.25  0.687  -0.55)'  constant/triSurface/halfParshall.org.stl
constant/triSurface/halfParshall.stl

runApplication blockMesh
runApplication extrudeMesh
runApplication surfaceFeatureExtract

runApplication decomposePar
mv log.decomposePar log.decomposePar.1
runParallel snappyHexMesh 4 -overwrite

runApplication reconstructParMesh -constant
runApplication createPatch -overwrite
runApplication transformPoints -translate '(4.25 -0.687 0.55)'
runApplication checkMesh -constant
runApplication changeDictionary -enableFunctionEntries
```

Mesh Generation (3/16)

Meaning of each step (1/4):

- **runApplication** and **runParallel** – these are auxiliary script functions, for logging the execution of the application.
- **surfaceTransformPoints** – used for centring the geometry onto the world referential.
- **blockMesh** – generates the base mesh, which wraps our geometry within it, acting as a bounding box. Requires the file “constant/polyMesh/blockMeshDict”.
- **extrudeMesh** – in our case, we use it to add one additional cell layer around the original base mesh, for improving the wrapping around our geometry. Requires the file “system/extrudeMeshDict”.

Mesh Generation (4/16)

Meaning of each step (2/4):

- **surfaceFeatureExtract** – will calculate the feature edges on our STL file. Requires these files:
  - "system/surfaceFeatureExtractDict"
  - "constant/triSurface/halfParshall.stl"
- **decomposePar** – will decompose our existing mesh so far into 4 subdomains, so that we can mesh with 4 processes in parallel. Requires the file "system/decomposeParDict".
- **mv log.decomposePar log.decomposePar.1** – this is a bureaucratic step, where we rename the log file that was generated on the previous step.

Mesh Generation (5/16)

Meaning of each step (3/4):

- **snappyHexMesh** – this is the main mesh generator we will use, which takes the base mesh we created and it will:
  - refine the mesh accordingly to our settings;
  - remove the cells that don't matter from the mesh, in this case, the cells that are outside of our geometry;
  - morph and cut (i.e. *snap*) the mesh's surface onto the surfaces of our geometry.

  All of the above settings are defined in the file "system/snappyHexMeshDict".
- **reconstructParMesh** – this will reconstruct the resulting 4 subdomain meshes into a single mesh.

Mesh Generation (6/16)

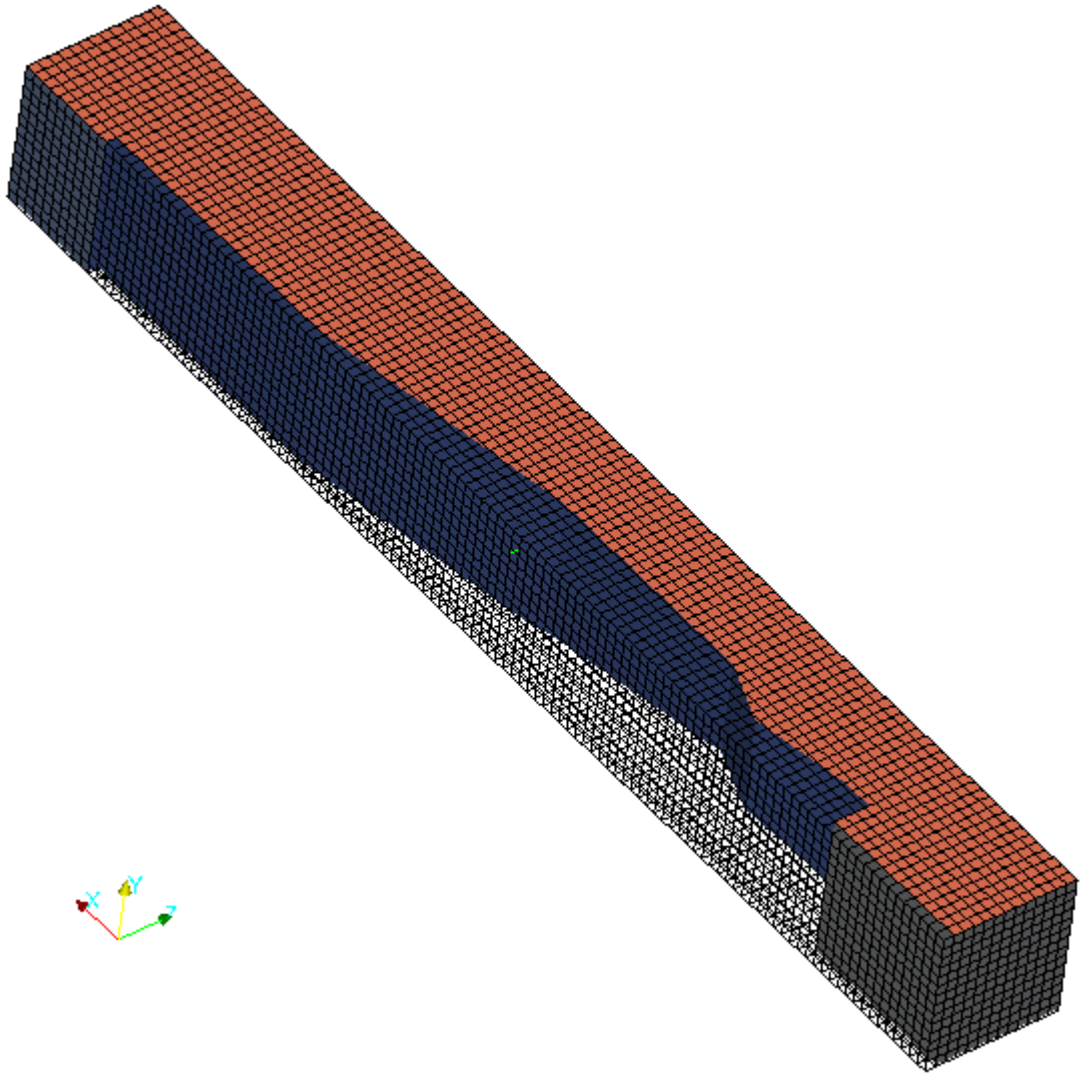
Meaning of each step (4/4):

- **createPatch** – it will clean up the list of patches in our mesh, because the original patches from the base mesh would otherwise remain present, with 0 faces assigned.
- **transformPoints** – used in our case for moving the whole mesh back into the original position of the original geometry.
- **checkMesh** – used for keeping a record of the characteristics of the mesh, including any diagnosed flaws.
- **changeDictionary** – in our case we use it for changing the type of surface boundary we want for each patch, namely if each surface is a “patch”, “wall” or “symmetry”.

Mesh Generation (7/16)

Visually seeing the mesh (1/5) - mesh done with **blockMesh**:

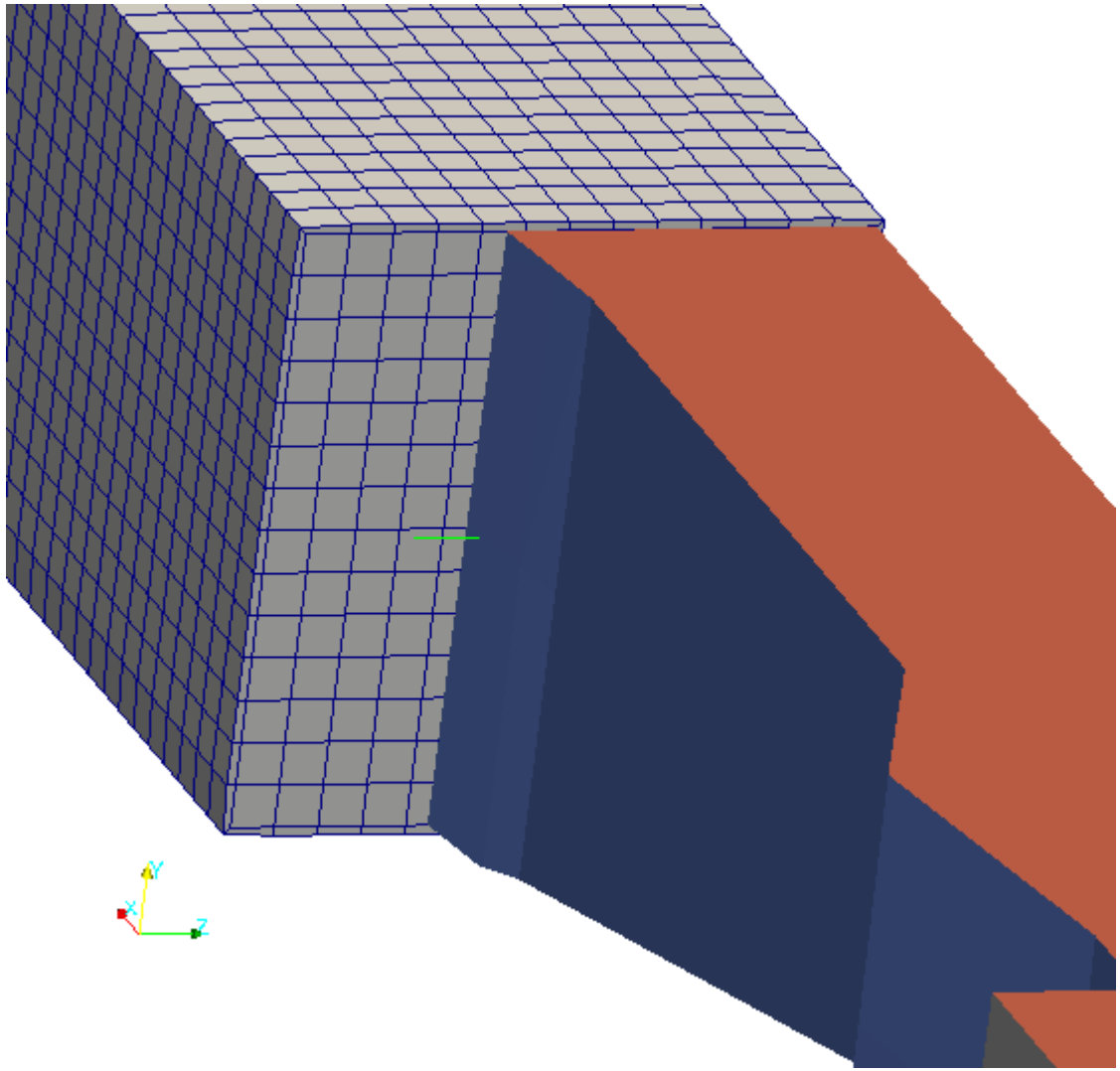
As you can see, it's a somewhat tight bounding box around our geometry, which is why we refer to this as the "background mesh".



Mesh Generation (8/16)

Visually seeing the mesh (2/5) - the extruded mesh:

This is a detail view of the inside of the mesh after the extrusion is done. This will make it easier for **snappyHexMesh** to *see* the geometry.

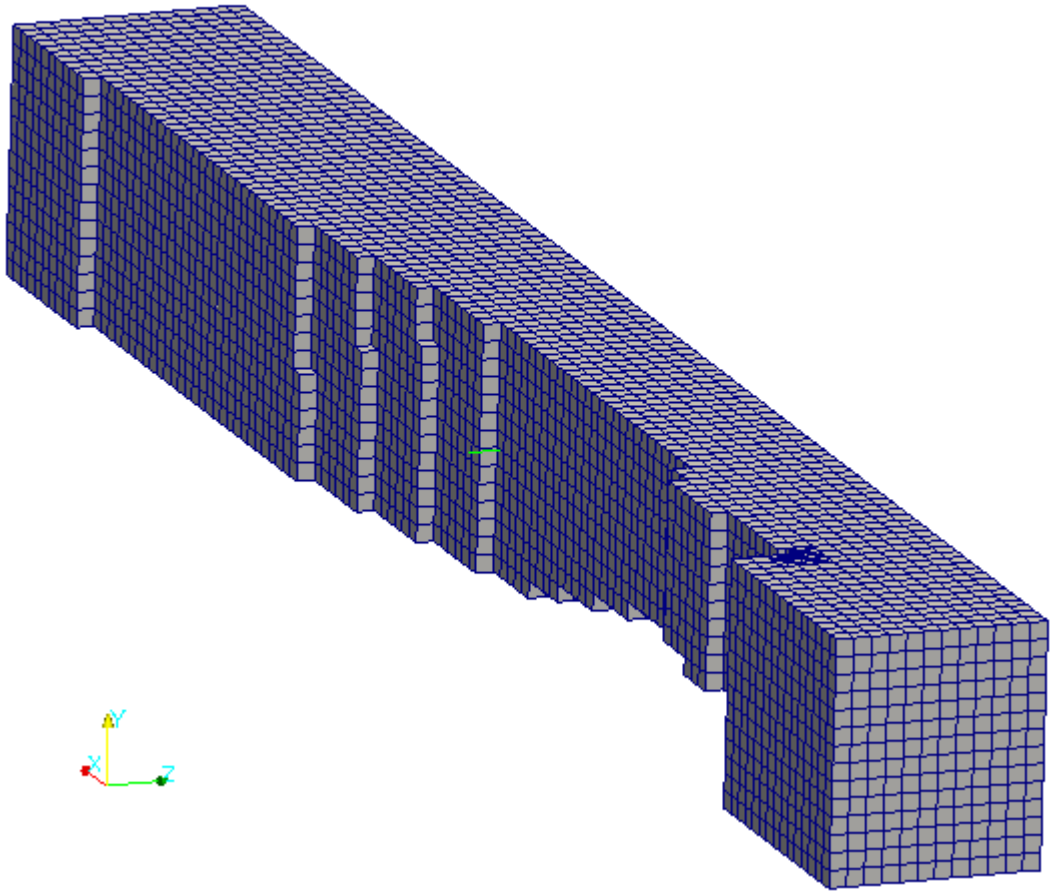


Mesh Generation (9/16)

Visually seeing the mesh (3/5) – 1st step of **snappyHexMesh**:

This is the result of the *castellation* step, where it:

1. Refines the mesh where asked to.
2. Removes the cells that are irrelevant for our final mesh.

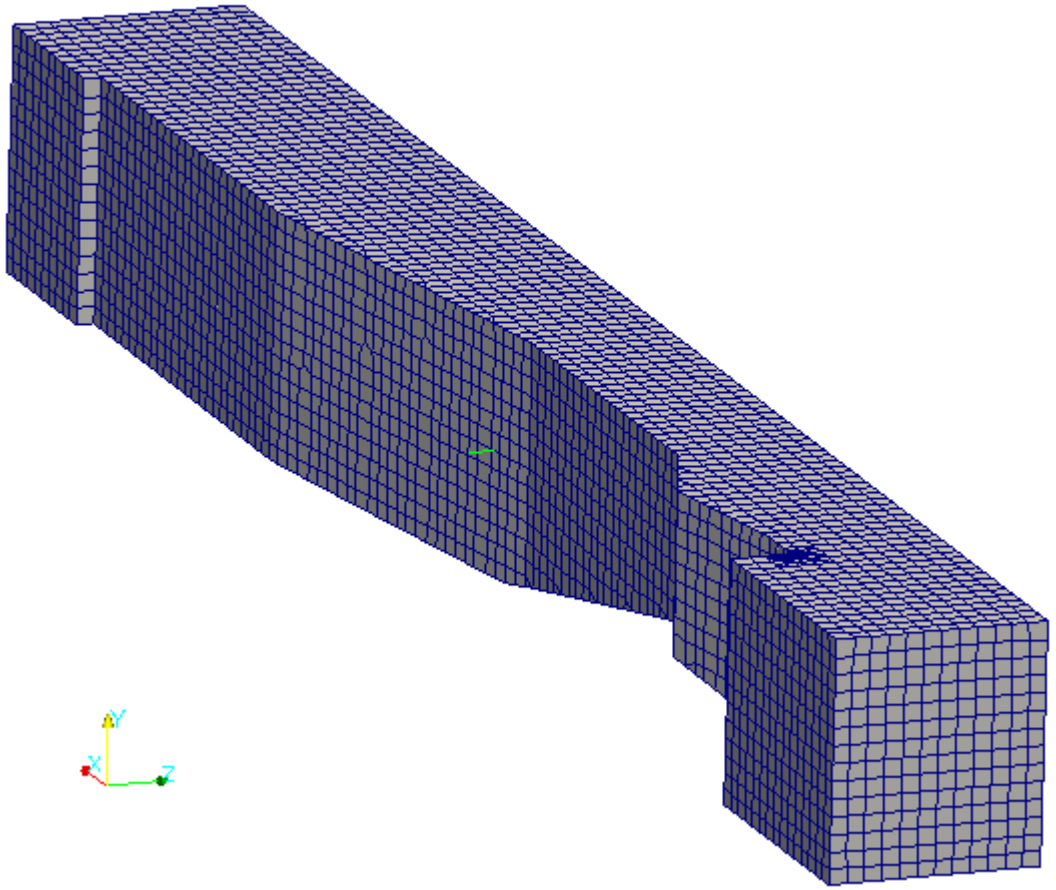


Mesh Generation (10/16)

Visually seeing the mesh (4/5) – 2nd step:

This is the result of the *snap* step, where it:

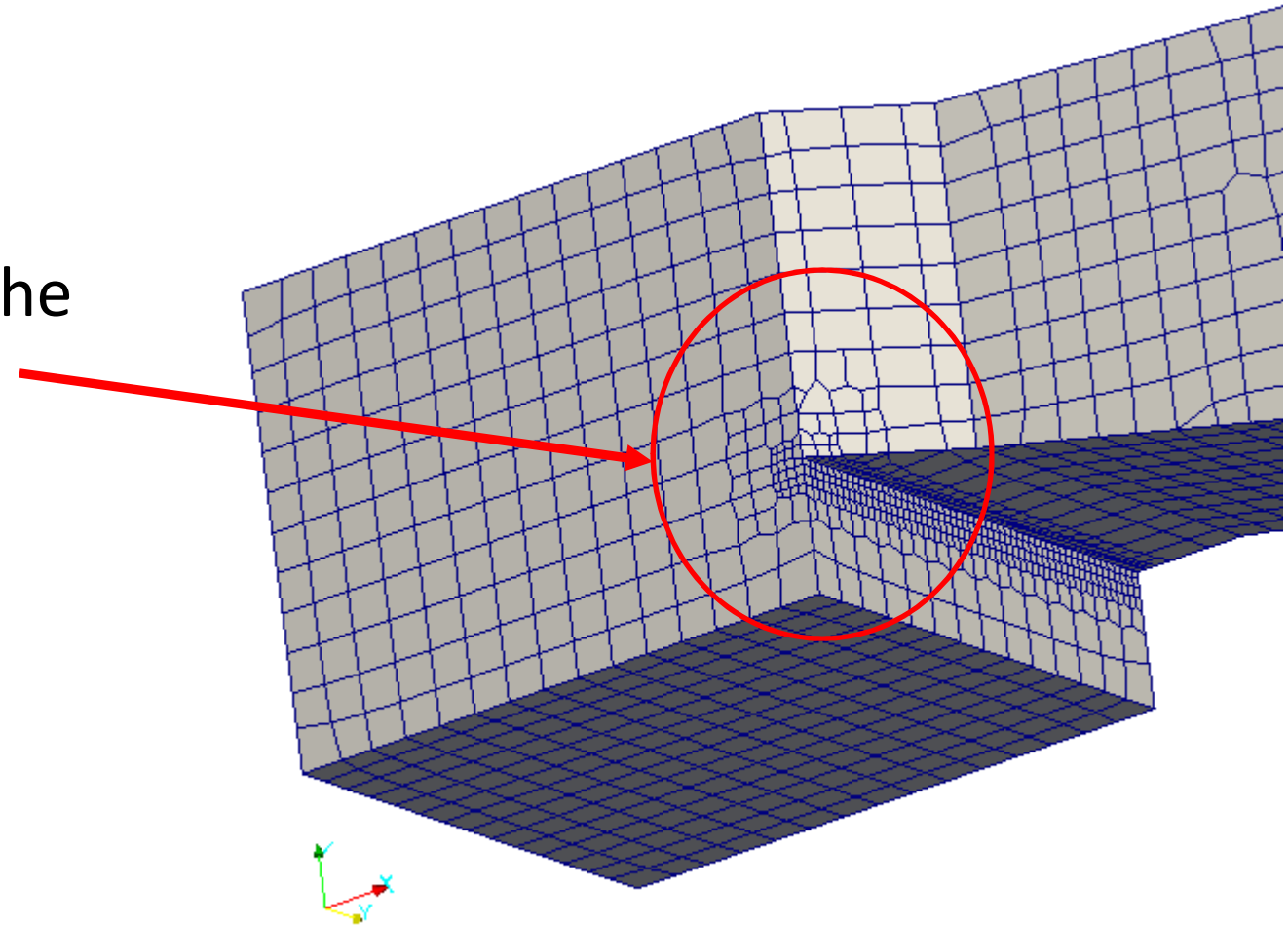
1. Cuts cells that overlap the geometry.
2. Morphs (snaps) the mesh onto the surface.



Mesh Generation (11/16)

Visually seeing the mesh (5/5) – refinement detail:

This is why we needed more refinement near the outlet.



Mesh Generation (12/16)

Note: This header is common to all of OpenFOAM's dictionary files:

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

It's part of OpenFOAM's open file format standard, so that special data readers aren't needed for human manipulation.

Mesh Generation (13/16)

Highlights of **blockMeshDict**:

```
convertToMeters 1;

vertices
(
    (-5.75 -0.686883 -0.65)
    (5.75  -0.686883 -0.65)
    (5.75   0.687 -0.65)
    (-5.75  0.687 -0.65)
    (-5.75 -0.686883 0.65)
    (5.75  -0.686883 0.65)
    (5.75   0.687 0.65)
    (-5.75  0.687 0.65)
);

blocks
(
    hex (0 1 2 3 4 5 6 7)
    (115 14 13) simpleGrading (1 1 1)
);
```

```
patches
(
    patch maxX
    ( (1 2 6 5) )
    patch minX
    ( (0 4 7 3) )
    patch maxY
    ( (3 7 6 2) )
    patch minY
    ( (1 5 4 0) )
    patch maxZ
    ( (4 5 6 7) )
    patch minZ
    ( (0 3 2 1) )
);
```

Mesh Generation (14/16)

Highlights of **snappyHexMeshDict** (1/3):

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

geometry
{
  "halfParshall.stl"
  {
    type triSurfaceMesh;
    regions
    {
      backWall
      {
        name backWall;
      }

       […]
```

```
      […]

      top
      {
        name top;
      }
    }
  }

  refinementBox
  {
   type     searchableBox;
   min      (-4.35 -1 -0.30);
   max      (-4.20  1 -0.15);
  }

};
```

Mesh Generation (15/16)

Highlights of **snappyHexMeshDict** (2/3):

```
// Settings for the castellatedMesh generation.
castellatedMeshControls
{ […]
    features
    ( {
        file "halfParshall.eMesh";
        level 0;
      } );

    refinementSurfaces
    {
      "halfParshall.stl"
      {
        level (0 0);
        regions
        {
          backWall
          {
            level (0 0);
```

Mesh Generation (16/16)

Highlights of **snappyHexMeshDict** (3/3):

```
  refinementRegions
  {
    refinementBox
    {
      mode inside;
      levels ((1e-15 2));
    }
  }

  locationInMesh (-1.212134e+000 8.290353e-003 -4.588275e-002);
}

// Settings for the snapping.
snapControls
{
  nSmoothPatch 3;
  tolerance 1.0;
  nSolveIter 30;
  nRelaxIter 5;
```

Preprocessing (1/9)

Model Properties (1/2)

These are defined in the file “constant/transportProperties”:

```
transportModel  Newtonian;
phases (water air);
water
{
    transportModel  Newtonian;
    nu              nu    [ 0 2 -1 0 0 0 0 ] 1.131131e-006;
    rho             rho   [ 1 -3 0 0 0 0 0 ] 9.990000e+002;
    mu              mu    [ 1 -1 -1 0 0 0 0 ] 1.130000e-003;
}
air
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ]  1.572e-05;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1.18;
    mu              mu    [ 1 -1 -1 0 0 0 0 ] 1.855e-05;
}
sigma           sigma [ 1 0 -2 0 0 0 0 ] 0;
```

Preprocessing (2/9)

Model Properties (2/2)

Where:

- *nu* – kinematic viscosity (m^2/s)
- *mu* – dynamic viscosity (kg.m/s)
- *rho* - volumetric mass density (kg/m^3)
- *sigma* - surface tension (kg/s^2 or N/m)
- *phases* – the list of named phases present in the domain.
  - The names do not strictly define the fluid their representing, they are only for identification purposes.

Preprocessing (3/9)

Boundary Conditions (1/2)

In a nutshell:

- 6 field files: *U*, *alpha.water*, *epsilon*, *k*, *nut*, *p_rgh*
- 4 major groups of boundary conditions per field:
  - Inlet – assigned to the “inlet” surface
  - Outlet – assigned to the “outlet” and “top” surfaces
  - Wall – assigned to the “backWall”, “bottomWall”, “sideWall” surfaces
  - Symmetry – assigned to the “symmetry” surface

Preprocessing (4/9)

Boundary Conditions (2/2) – For example, *U* field file:

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0.0 0.0 0.0);
boundaryField
{
    backWall
    {
        type            fixedValue;
        value           uniform (0.0 0.0 0.0);
    }
…
    inlet
    {
      type                flowRateInletVelocity;
      massFlowRate        375;
      rho                 rho;
      rhoInlet            999.0;
    }
…
```

Preprocessing (5/9)

system/fvSolution (1/3):
This dictionary file was designed to handle the settings for the linear equation solvers and the algorithms to be used by a solver application, e.g. **interFoam**.

Starting with the linear equation solvers, these are configured inside block list:

```
solvers
{
  …
}
```

The next few slides show one example.

Preprocessing (6/9)

system/fvSolution (2/3):

For configuring the linear equation solvers for the fields that start with “alpha.water”, the example case we are using has the following settings:

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

    MULESCorr       yes;
    nLimiterIter    3;

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

Preprocessing (7/9)

system/fvSolution (3/3):

Regarding the algorithm, **interFoam** uses PIMPLE, defined at the same levels as “solvers”:

```
solvers
{
  …
}

PIMPLE
{
  momentumPredictor   no;
  nOuterCorrectors    1;
  nCorrectors         3;
  nNonOrthogonalCorrectors 1;
}
```

Preprocessing (8/9)

system/fvSchemes (1/2):

Finite volume discretization schemes:

```
ddtSchemes
{
  default  Euler;
}

gradSchemes
{
  default  Gauss linear;
}

laplacianSchemes
{
  default  Gauss linear corrected;
}
```

```
interpolationSchemes
{
  default   linear;
}

snGradSchemes
{
  default   corrected;
}

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

Preprocessing (9/9)

system/fvSchemes (2/2):

These usually come after “gradSchemes”:

```
divSchemes
{
  default                         none;
  div(rhoPhi,U)      Gauss upwind;
  div(phi,alpha)     Gauss upwind;
  div(phirb,alpha)   Gauss upwind;
  div(phi,k)         Gauss upwind;
  div(phi,epsilon)   Gauss upwind;
  div((muEff*dev(T(grad(U))))) Gauss linear;
}
```

Simulation (1/4)

Simulation steps overview – as simple as looking into the contents of the **Allrun** script:

```
./Allrun.pre

cp -r 0.org 0

runApplication decomposePar -force
mv log.decomposePar log.decomposePar.2

runParallel renumberMesh 4 -overwrite
#runParallel setFields 4
runParallel interFoam 4

runApplication reconstructPar
```

In the next slides we will see what each does…

Simulation (2/4)

Simulation steps (1/3):

- **Allrun.pre** – We learned about it in the meshing section.
- **cp -r 0.org 0** – Deploy initial fields for the time step “0”.
- **decomposePar -force** – This will re-decompose our mesh, along with the fields. We could have used the option “**-fields**”, but we want to make sure that the mesh is well balanced, which might not be the case when *snappyHexMesh* is finished.
- **renumberMesh** – This application is meant to optimize how the cells (and respective data) in the mesh are organized, so that the equation matrices have a diagonal bandwidth as small as possible. Performance improvements can reach 30% less runtime.

Simulation (3/4)

Simulation steps (2/3):

- **setFields** – Not used in our example case, but this is one of the reasons as to why we need the folder “0.org” to be created separately, since running *setFields* will change the field files in the “0” folder.
- **interFoam** – This is the solver used in our example case. Quoting from the source code:

Solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach.
The momentum and other fluid properties are of the "mixture" and a single momentum equation is solved.
Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

Simulation (4/4)

Simulation steps (3/3):

- **reconstructPar** – For reconstructing the time steps that were generated while running in parallel.

Beyond this, comes the need for monitoring the output of the log file… which will seem fairly cryptic at first glance. For example:

```
GAMG:  Solving for p_rgh, Initial residual = 1, Final residual
= 0.020995727, No Iterations 4
```

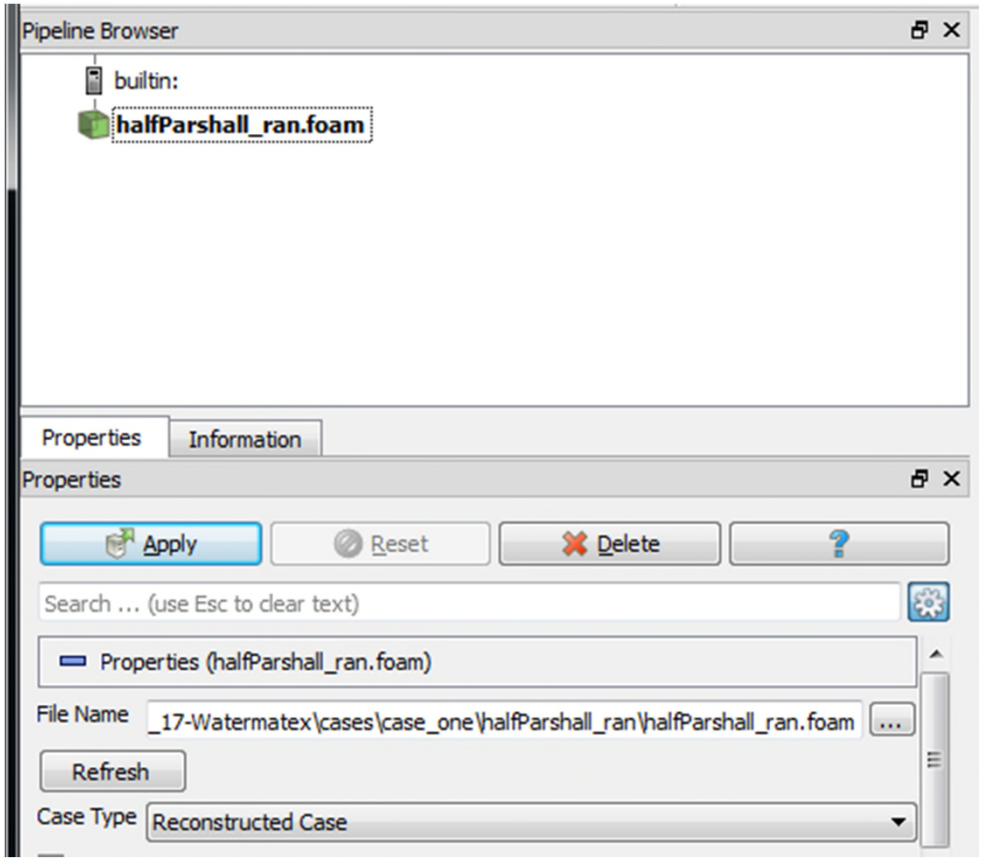
Not to worry, there is more than one way to plot the values that matter to us.

Post-processing (1/3)

The easiest post-processing is done by using ParaView, which can be launched with the script:

```
paraFoam
```

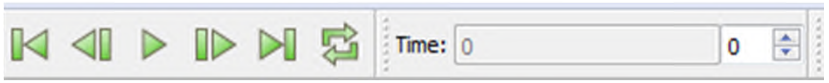
ParaView will start and show on the left side of the window, something similar to the one on the right.
Clicking on the “Apply” button will load the case.



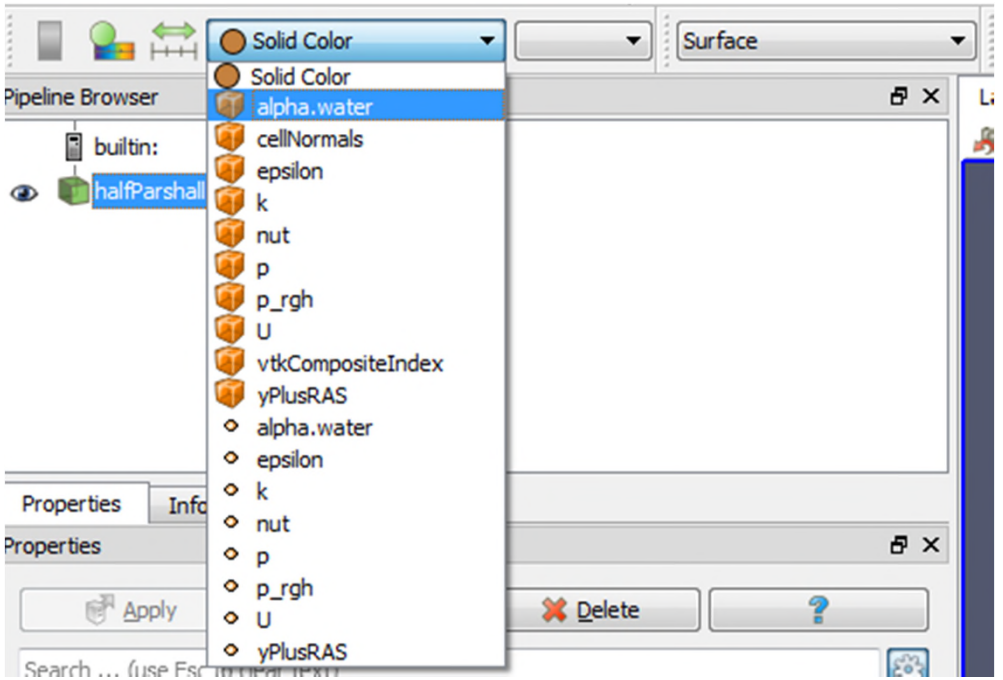
Post-processing (2/3)

The mouse controls with the 3D view are similar to most 3D CAD software, although the actions done by each button may be switched.

These are the time controls:

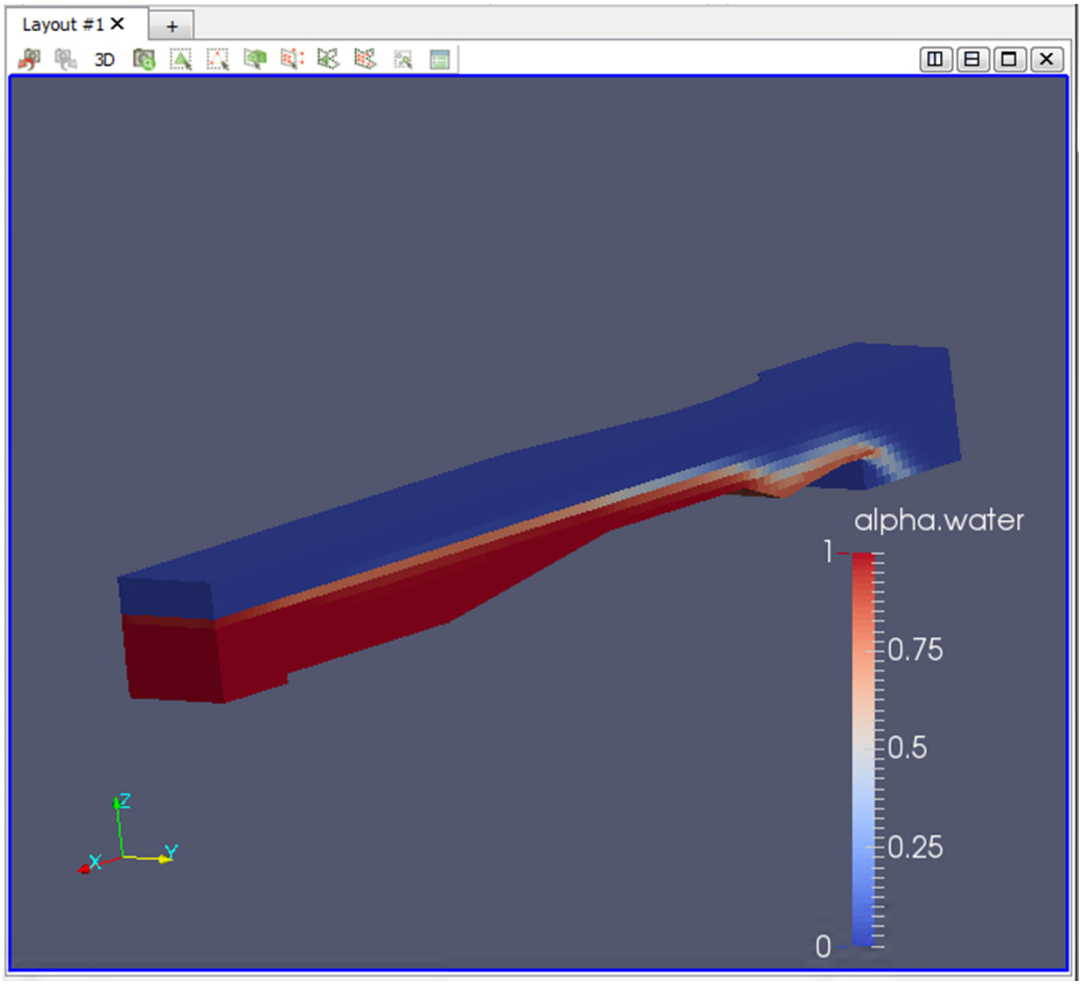


And this is an example on how to change between fields to be rendered.



Post-processing (3/3)

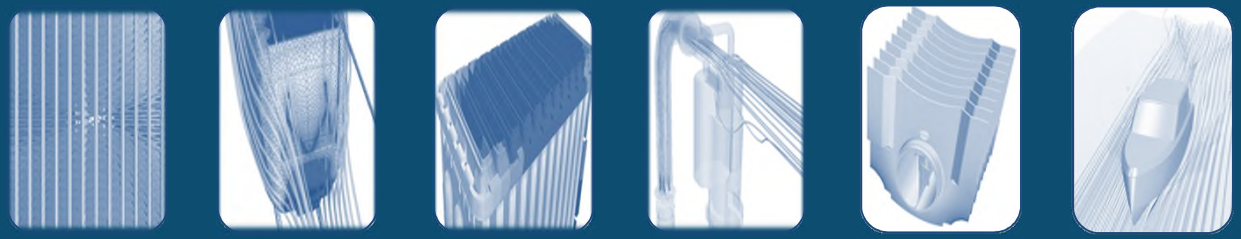
By choosing “alpha.water” and turning on the legend, we can see the result below:



The legend can be moved with the mouse.

Thank you for your time.

Next:
3 - Meshing



bluecape
Computer Applications
in Science & Engineering