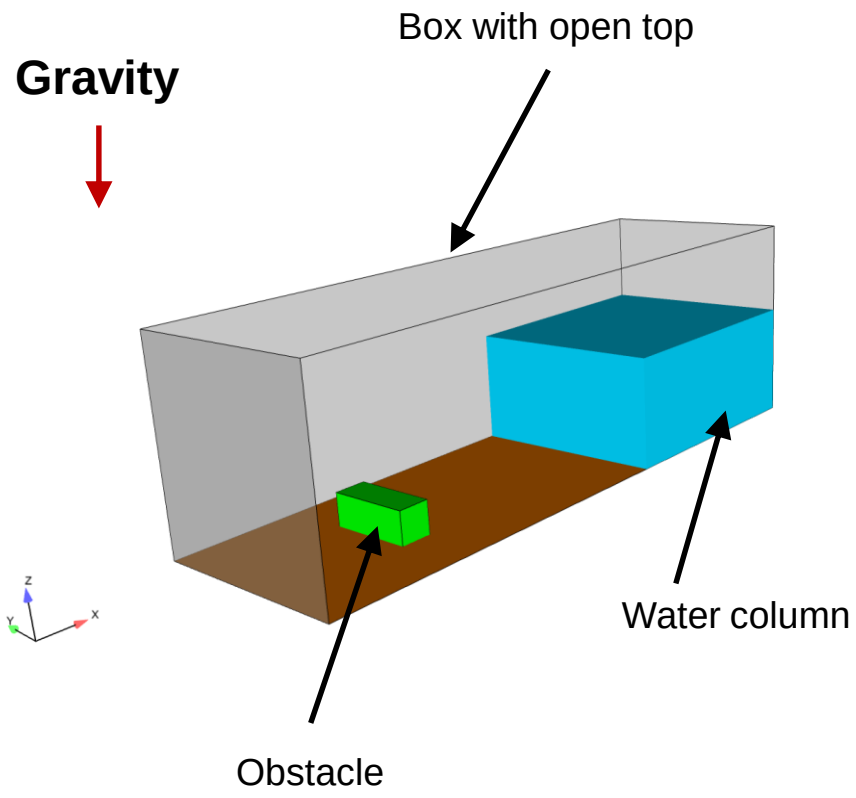
3D Dam break – Free surface flow

Dam break free surface flow



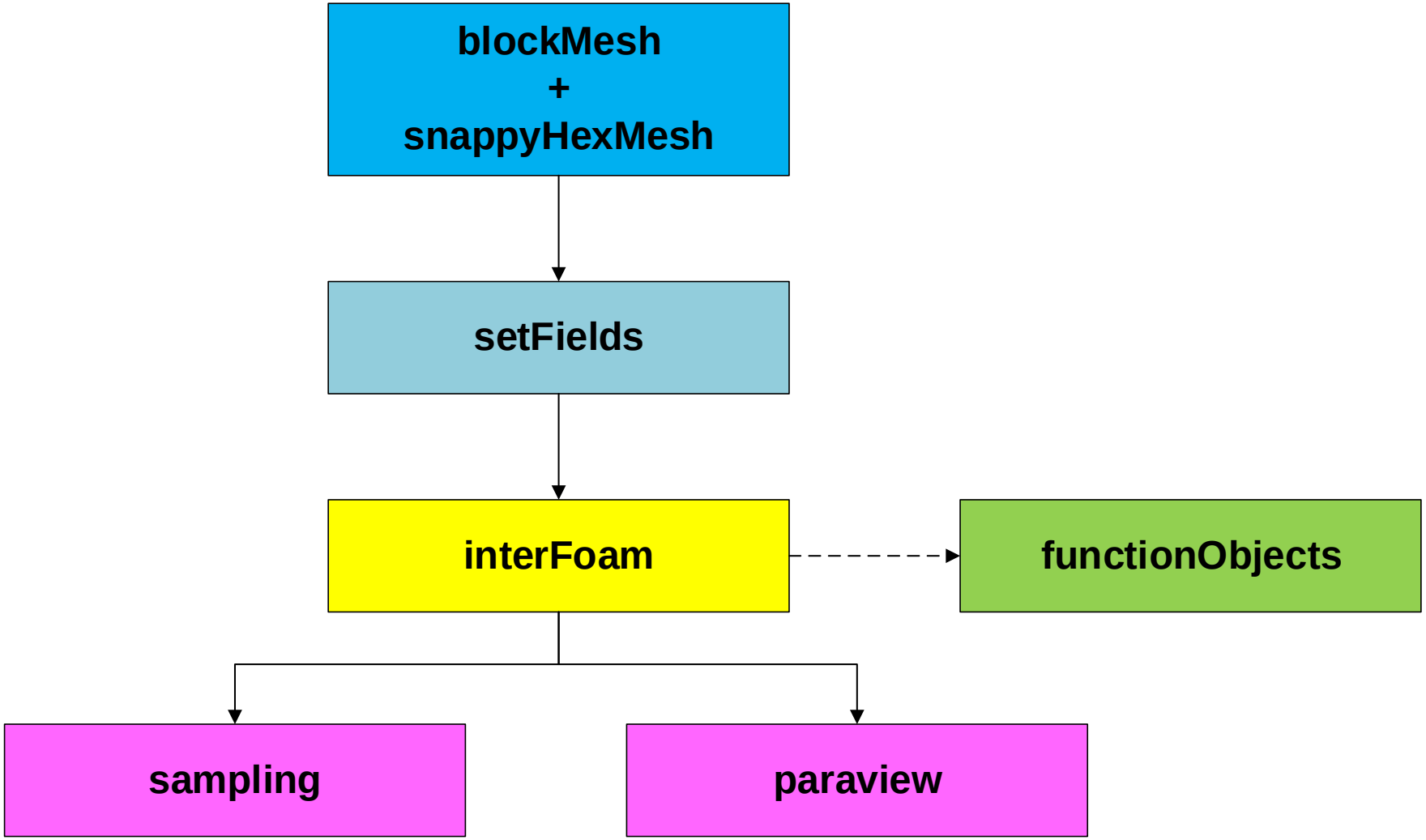
Physical and numerical side of the problem:

- In this case we are going to use the volume of fluid (VOF) method.
- This method solves the incompressible Navier-Stokes equations plus an additional equation to track the phases (free surface location).
- As this is a multiphase case, we need to define the physical properties for each phase involved (viscosity, density and surface tension).
- The working fluids are water and air.
- Additionally, we need to define the gravity vector and initialize the two flows.
- This is a three-dimensional and unsteady case.
- The details of the case setup can be found in the following reference:

  A Volume-of-Fluid Based Simulation Method for Wave Impact Problems.
  Journal of Computational Physics 206(1):363-393.
  June, 2005.

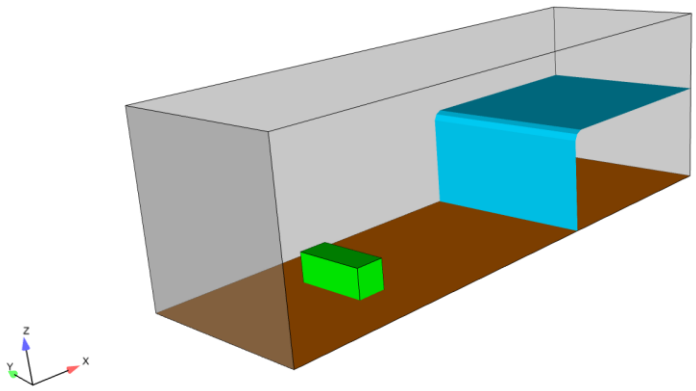
3D Dam break – Free surface flow

Workflow of the case

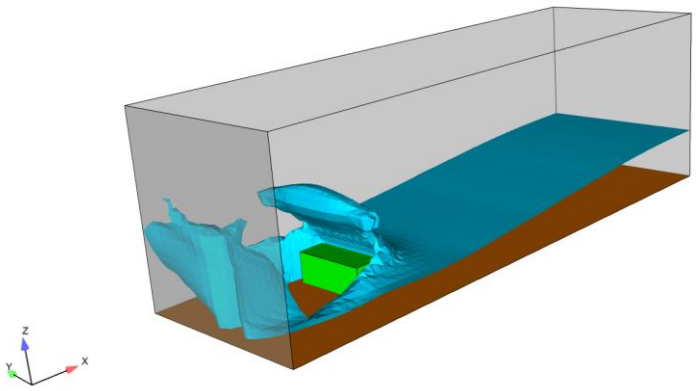


3D Dam break – Free surface flow

At the end of the day, you should get something like this



Initial conditions – Coarse mesh

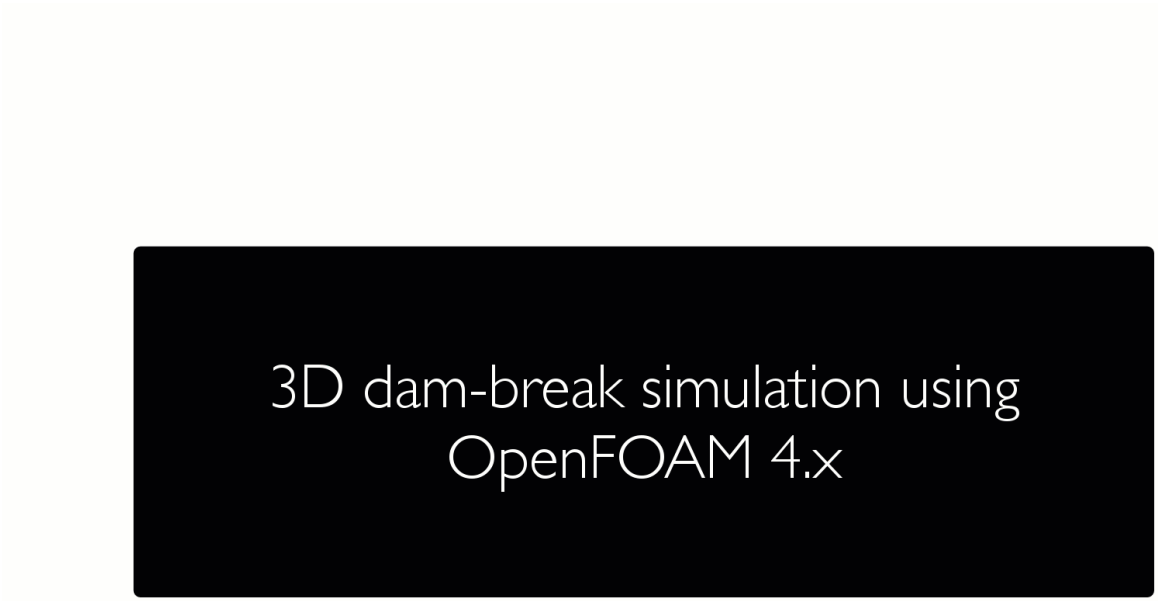


Solution at Time = 1 second – Coarse mesh

3D Dam break – Free surface flow

VOF Fraction (Free surface tracking) – Very fine mesh

http://www.wolfdynamics.com/validations/3d_db/dbbreak.gif



3D Dam break – Free surface flow

- Let us run this case. Go to the directory:

```
$PTOFC/3d_damBreak
```

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

3D Dam break – Free surface flow

What are we going to do?

- We will use this case to introduce the multiphase solver `interFoam`.
- `interFoam` is a solver for 2 incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach
- We will define the physical properties of two phases and we are going to initialize these phases.
- We will define the gravity vector in the dictionary `g`.
- After finding the solution, we will visualize the results. This is an unsteady case so now we are going to see things moving.
- We are going to briefly address how to post-process multiphase flows.
- We are going to generate the mesh using snappyHexMesh, but for the purpose of this tutorial we are not going to discuss the dictionaries.
- Remember, different solvers have different input dictionaries.

3D Dam break – Free surface flow

The **`constant`** directory

- In this directory, we will find the following compulsory dictionary files:
  - *`g`*
  - *`transportProperties`*
  - *`turbulenceProperties`*

- *`g`* contains the definition of the gravity vector.
- *`transportProperties`* contains the definition of the physical properties of each phase.
- *`turbulenceProperties`* contains the definition of the turbulence model to use.

3D Dam break – Free surface flow

The g dictionary file

```
8     FoamFile
9     {
10        version     2.0;
11        format      ascii;
12        class       uniformDimensionedVectorField;
13        location    "constant";
14        object      g;
15    }
17
18    dimensions      [0 1 -2 0 0 0 0];
19    value           (0 0 -9.81);
```

- This dictionary file is located in the directory **`constant`**.
- For multiphase flows, this dictionary is compulsory.
- In this dictionary we define the gravity vector (line 19).
- Pay attention to the **class** type (line 12).

3D Dam break – Free surface flow

The *transportProperties* dictionary file

Primary phase

```
17    phases (water air);
18
19    water
20    {
21        transportModel  Newtonian;
22        nu              [0 2 -1 0 0 0 0] 1e-06;
23        rho             [1 -3 0 0 0 0 0] 1000;
24    }
25
26    air
27    {
28        transportModel  Newtonian;
29        nu              [0 2 -1 0 0 0 0] 1.48e-05;
30        rho             [1 -3 0 0 0 0 0] 1;
31    }
32
33    sigma           [1 0 -2 0 0 0 0] 0.07;
```

- This dictionary file is located in the directory **`constant`**.
- We first define the name of the phases (line 17). In this case we are defining the names **water** and **air**. The first entry in this list is the primary phase (**water**).
- The name of the primary phase is the one you will use to initialize the solution.
- The name of the phases is given by the user.
- In this file we set the kinematic viscosity (**nu**), density (**rho**) and transport model (**transportModel**) of the phases.
- We also define the surface tension (**sigma**).

3D Dam break – Free surface flow

The *turbulenceProperties* dictionary file

- In this dictionary file we select what model we would like to use (laminar or turbulent).
- This dictionary is compulsory.
- In this case we use a RANS turbulence model (kEpsilon).

```
17    simulationType    RAS;
18
19    RAS
20    {
21        RASModel kEpsilon;
22
23        turbulence on;
24
25        printCoeffs on;
26    }
```

3D Dam break – Free surface flow

The **0** directory

- In this directory, we will find the dictionary files that contain the boundary and initial conditions for all the primitive variables.
- As we are solving the incompressible RANS Navier-Stokes equations using the VOF method, we will find the following field files:

  - *alpha.water* (volume fraction of water phase)
  - *p_rgh* (pressure field minus hydrostatic component)
  - *U* (velocity field)
  - *k* (turbulent kinetic energy field)
  - *epsilon* (rate of dissipation of turbulence energy field)
  - *nut* (turbulence viscosity field)

3D Dam break – Free surface flow

The file *0/alpha.water*

```
17  dimensions      [0 0 0 0 0 0 0];
18
19  internalField   uniform 0;
20
21  boundaryField
22  {
23      front
24      {
25          type            zeroGradient;
26      }
27      back
28      {
29          type            zeroGradient;
30      }
31      left
32      {
33          type            zeroGradient;
34      }
35      right
36      {
37          type            zeroGradient;
38      }
39      bottom
40      {
41          type            zeroGradient;
42      }
43      top
44      {
45          type            inletOutlet;
46          inletValue      uniform 0;
47          value           uniform 0;
48      }
49      stlSurface
50      {
51          type            wall;
52      }
53
54  }
```

- This file contains the boundary and initial conditions for the non-dimensional scalar field **alpha.water**
- This file is named *alpha.water*, because the primary phase is water (we defined the primary phase in the *transportProperties* dictionary).
- Initially, this field is initialize as 0 in the whole domain (line 19). This means that there is no water in the domain at time 0. Later, we will initialize the water column and this file will be overwritten with a non-uniform field for the **internalField**.
- For the **front**, **back**, **left**, **right**, **bottom** and **stlSurface** patches we are using a **zeroGradient** boundary condition (we are just extrapolating the internal values to the boundary face).
- For the **top** patch we are using an **inletOutlet** boundary condition. This boundary condition avoids backflow into the domain. If the flow is going out it will use **zeroGradient** and if the flow is coming back it will assign the value set in the keyword **inletValue** (line 46).

3D Dam break – Free surface flow

The file *0/p_rgh*

```
17    dimensions      [1 -1 -2 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        front
24        {
25            type            fixedFluxPressure;
26            value           uniform 0;
27        }
28        back

33        left

38        right

43        bottom

48        top
49        {
50            type            totalPressure;
51            p0              uniform 0;
52            U               U;
53            phi             phi;
54            rho             rho;
55            psi             none;
56            gamma           1;
57            value           uniform 0;
58        }
59        stlSurface
60        {
61            type            fixedFluxPressure;
62            value           uniform 0;
63        }
64
65    }
```

- This file contains the boundary and initial conditions for the dimensional scalar field **p_rgh**. The dimensions of this field are given in Pascal (line 17)
- This scalar field contains the value of the static pressure field minus the hydrostatic component.
- This field is initialize as 0 in the whole domain (line 19).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using a **fixedFluxPressure** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **totalPressure** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).

3D Dam break – Free surface flow

The file *0/U*

```
17    dimensions      [0 -1 -1 0 0 0 0];
18
19    internalField   uniform (0 0 0);
20
21    boundaryField
22    {
23        front
24        {
25            type            fixedValue;
26            value           uniform (0 0 0);
27        }
28        back

33        left

38        right

43        bottom

48        top
49        {
50            type            pressureInletOutletVelocity;
51            value           uniform (0 0 0);
52        }
53        stlSurface
54        {
55            type            fixedValue;
56            value           uniform (0 0 0);
57        }
58
59    }
```

- This file contains the boundary and initial conditions for the dimensional vector field **U**.
- We are using uniform initial conditions and the numerical value is **(0 0 0)** (keyword **internalField** in line 19).
- The **front**, **back**, **left, right, bottom** and **stlSurface** patches are no-slip walls, therefore we impose a **fixedValue** boundary condition with a value of **(0 0 0)** at the wall.
- For the **top** patch we are using the **pressureInlterOutletVelocity** boundary condition (refer to the source code or doxygen documentation to know more about this boundary condition).

3D Dam break – Free surface flow

The file *0/k*

```
17    dimensions      [0 2 -2 0 0 0 0];
18
19    internalField   uniform 0.1;
20
21    boundaryField
22    {
23        "(front|back|left|right|bottom|stlSurface)"
24        {
25            type            kqRWallFunction;
26            value           $internalField;
27        }
28
29        top
30        {
31            type            inletOutlet;
32            inletValue      $internalField;
33            value           $internalField;
34        }
35
36    }
```

- This file contains the boundary and initial conditions for the dimensional scalar field **k**.
- This scalar (turbulent kinetic energy), is related to the turbulence model.
- This field is initialize as 0.1 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using a **kqRWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **inletOutlet** boundary condition, this boundary condition handles backflow (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The file *0/epsilon*

```
17    dimensions      [0 2 -3 0 0 0 0];
18
19    internalField   uniform 0.1;
20
21    boundaryField
22    {
23        "(front|back|left|right|bottom|stlSurface)"
24        {
25            type            epsilonWallFunction;
26            value           $internalField;
27        }
28
29        top
30        {
31            type            inletOutlet;
32            inletValue      $internalField;
33            value           $internalField;
34        }
35
36    }
```

- This file contains the boundary and initial conditions for the dimensional scalar field **epsilon**.
- This scalar (rate of dissipation of turbulence energy), is related to the turbulence model.
- This field is initialize as 0.1 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using a **epsilonWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **inletOutlet** boundary condition, this boundary condition handles backflow (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The file *0/nut*

```
17    dimensions      [0 2 -1 0 0 0 0];
18
19    internalField   uniform 0;
20
21    boundaryField
22    {
23        "(front|back|left|right|bottom|stlSurface)"
24        {
25            type            nutkWallFunction;
26            value           $internalField;
27        }
28
29        top
30        {
31            type            calculated;
32            value           $internalField;;
33        }
34
35    }
```

- This file contains the boundary and initial conditions for the dimensional scalar field **nut**.
- This scalar (turbulent viscosity), is related to the turbulence model.
- This field is initialize as 0 in the whole domain, and all the boundary patches take the same value (**$internalField**).
- For the **front**, **back**, **left, right, bottom** and **stlSurface** patches we are using a **nutkWallFunction** boundary condition, which applies a wall function at the walls (refer to the source code or doxygen documentation to know more about this boundary condition).
- For the **top** patch we are using the **calculated** boundary condition, this boundary condition computes the value of nut from k and epsilon (refer to the source code or doxygen documentation to know more about this boundary condition).
- We will deal with turbulence modeling later.

3D Dam break – Free surface flow

The **`system`** directory

- The **`system`** directory consists of the following compulsory dictionary files:
  - *`controlDict`*
  - *`fvSchemes`*
  - *`fvSolution`*

- *`controlDict`* contains general instructions on how to run the case.
- *`fvSchemes`* contains instructions for the discretization schemes that will be used for the different terms in the equations.
- *`fvSolution`* contains instructions on how to solve each discretized linear equation system.

3D Dam break – Free surface flow

The *controlDict* dictionary

```
17  application     interFoam;
18
19  startFrom       startTime;
20
21  startTime       0;
22
23  stopAt          endTime;
24
25  endTime         8;
26
27  deltaT          0.0001;
28
29  writeControl    adjustableRunTime;
30
31  writeInterval   0.02;
32
33  purgeWrite      0;
34
35  writeFormat     ascii;
36
37  writePrecision  8;
38
39  writeCompression uncompressed;
40
41  timeFormat      general;
42
43  timePrecision   8;
44
45  runTimeModifiable yes;
46
47  adjustTimeStep  yes;
48
49  maxCo           0.5;
50  maxAlphaCo      0.5;
51  maxDeltaT       0.01;
```

- This case starts from time 0 (**startTime**), and it will run up to 8 seconds (**endTime**).
- The initial time step of the simulation is 0.0001 seconds (**deltaT**).
- It will write the solution every 0.02 seconds (**writeInterval**) of simulation time (**runTime**). It will automatically adjust the time step (**adjustableRunTime**), in order to save the solution at the precise write interval.
- It will keep all the solution directories (**purgeWrite**).
- It will save the solution in ascii format (**writeFormat**).
- The write precision is 8 digits (**writePrecision**). It will only save eight digits in the output files.
- And as the option **runTimeModifiable** is on, we can modify all these entries while we are running the simulation.
- In line 47 we turn on the option **adjustTimeStep**. This option will automatically adjust the time step to achieve the maximum desired courant number (lines 49-50). We also set a maximum time step in line 51.
- Remember, the first time step of the simulation is done using the value set in line 27 and then it is automatically scaled to achieve the desired maximum values (lines 49-51).

3D Dam break – Free surface flow

The *controlDict* dictionary

```
55      functions
56      {

60      minmaxdomain
61      {
62          type fieldMinMax;
63
64          functionObjectLibs ("libfieldFunctionObjects.so");
65
66          enabled true; //true or false
67
68          mode component;
69
70          writeControl timeStep;
71          writeInterval 1;
72
73          log true;
74
75          fields (p p_rgh U alpha.water k epsilon);
76      }

144     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 60-76 we define the **fieldMinMax functionObject** which computes the minimum and maximum values of the field variables (**p p_rgh U alpha.water k epsilon**).

3D Dam break – Free surface flow

The *controlDict* dictionary

```
55      functions
56      {

81      water_in_domain
82      {
83          type            volRegion;
84          functionObjectLibs ("libfieldFunctionObjects.so");
85          enabled         true;
86
87          enabled         true;
88
89          //writeControl     outputTime;
90          writeControl    timeStep;
91          writeInterval   1;
92
93          log             true;
94
95          regionType      all;
96
97         operation       volIntegrate;
98         fields
99         (
100             alpha.water
101         );
102     }

144     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 81-102 we define the **volRegion functionObject** which computes the volume integral (**volIntegrate**) of the field variable **alpha.water** in all the domain.
- Basically, we are monitoring the quantity of water in the domain.

3D Dam break – Free surface flow

The *controlDict* dictionary

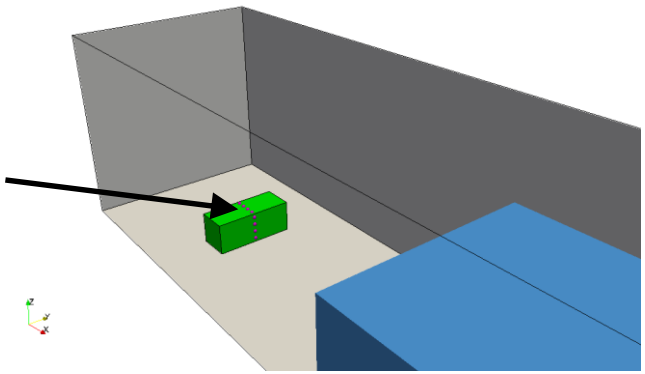
```
55      functions
56      {

107      probes1
108      {
109          type            probes;
110          functionObjectLibs ("libsampling.so");
111
112          probeLocations
113          (
114              (0.82450002 0 0.021)
115              (0.82450002 0 0.061)
116              (0.82450002 0 0.101)
117              (0.82450002 0 0.141)
118              (0.8035 0 0.161)
119              (0.7635 0 0.161)
120              (0.7235 0 0.161)
121              (0.6835 0 0.161)
122          );
123
124         fields
125          (
126              p p_rgh
127         );
128
129          writeControl    timeStep;
130          writeInterval   1;
131      }

144      };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 107-131 we define the **probes functionObject** which sample the selected fields (lines 124-127) at the selected locations (lines 112-122).
- This sampling is done on-the-fly. All the information sample by this **functionObject** is saved in the directory **`./postProcessing/probes1`**
- As we are sampling starting from time 0, the sampled data will be located in the directory:

  `postProcessing/probes1/0`

- Feel free to open the files located in the directory **`postProcessing/probes1/0`** using your favorite text editor.

Sampling locations (probeLocations)



3D Dam break – Free surface flow

The *controlDict* dictionary

```
55      functions
56      {

135      yplus
136      {
137          type            yPlus;
138          functionObjectLibs ("libutilityFunctionObjects.so ");
139          enabled true;
140          writeControl outputTime;
141      }

144     };
```

- Let us take a look at the **functionObjects** definitions.
- In lines 135-141 we define the **yplus functionObject** which computes the yplus value.
- This quantity is related to the turbulence modeling.
- This **functionObject** will save the yplus field in the solution directories with the same saving frequency as the solution (line 140).
- It will also save the minimum, maximum and mean values of yplus in the directory:

`postProcessing/yplus`

3D Dam break – Free surface flow

The *fvSchemes* dictionary

```
17    ddtSchemes
18    {
19        default         Euler;
21    }
22
23    gradSchemes
24    {
25        default         Gauss linear;
26        grad(U)         cellLimited Gauss linear 1;
27    }
28
29    divSchemes
30    {
31        div(rhoPhi,U)  Gauss linearUpwindV grad(U);
32        div(phi,alpha)  Gauss vanLeer;
33        div(phirb,alpha) Gauss linear;
35        div(phi,k) Gauss upwind;
36        div(phi,epsilon) Gauss upwind;
37        div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
38    }
39
40    laplacianSchemes
41    {
42        default         Gauss linear corrected;
43    }
44
45    interpolationSchemes
46    {
47        default         linear;
48    }
49
50    snGradSchemes
51    {
52        default         corrected;
53    }
```

- In this case, for time discretization (**ddtSchemes**) we are using the **Euler** method.
- For gradient discretization (**gradSchemes**) we are using the **Gauss linear** as the default method and slope limiters (**cellLimited**) for the velocity gradient or **grad(U)**.
- For the discretization of the convective terms (**divSchemes**) we are using **linearUpwindV** interpolation method for the term **div(rhoPhi,U)**.
- For the term **div(phi,alpha)** we are using **vanLeer** interpolation. For the term **div(phirb,alpha)** we are using **linear** interpolation. These terms are related to the volume fraction equation.
- For the terms **div(phi,alpha)** and **div(phi,alpha)** we are using upwind (these terms are related to the turbulence modeling).
- **For the term div(((rho*nuEff)*dev2(T(grad(U)))))** we are using **linear** interpolation (this term is related to the turbulence modeling).
- For the discretization of the Laplacian (**laplacianSchemes** and **snGradSchemes**) we are using the **Gauss linear corrected** method
- In overall, this method is second order accurate but a little bit diffusive. Remember, at the end of the day we want a solution that is second order accurate.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
17    solvers
18    {
19        "alpha.water.*"
20        {
21            nAlphaCorr      3;
22            nAlphaSubCycles 1;
23            cAlpha          1;
24
25            MULESCorr       yes;
26            nLimiterIter    10;
27
28            solver          smoothSolver;
29            smoother        symGaussSeidel;
30            tolerance       1e-8;
31            relTol          0;
32        }
33
34        "(pcorr|pcorrFinal)"
35        {
36            solver          PCG;
37            preconditioner  DIC;
38            tolerance       1e-8;
39            relTol          0;
40        }
41
42        p_rgh
43        {
44            solver          PCG;
45            preconditioner  DIC;
46            tolerance       1e-06;
47            relTol          0.01;
48            minIter         1;
49        }
```

- To solve the volume fraction or **alpha.water** (lines 19-32) we are using the **smoothSolver** method.
- In line 25 we turn on the semi-implicit method MULES. The keyword **nLimiterIter** controls the number of MULES iterations over the limiter.
- To have more stability it is possible to increase the number of loops and corrections used to solve **alpha.water** (lines 21-22).
- The keyword **cAlpha** (line 23) controls the sharpness of the interface (1 is usually fine for most cases).
- In lines 34-40 we setup the solver for **pcorr** and **pcorrFinal** (pressure correction).
- In this case **pcorr** is solved only one time at the beginning of the computation.
- In lines 42-49 we setup the solver for **p_rgh**.
- The keyword **minIter** (line 48), means that the linear solver will do at least one iteration.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
51        p_rghFinal
52        {
53            $p_rgh;
54            relTol          0;
55            minIter         1;
56        }
57
58        "(U|UFinal)"
59        {
60            solver          PBiCGStab;
61            Preconditioner  DILU;
62            tolerance       1e-08;
63            relTol          0;
72        }
73
74        "(k|epsilon).*"
75        {
76            solver          PBiCGStab;
77            Preconditioner  DILU;
78            tolerance       1e-08;
79            relTol          0;
80        }
81    }
82
```

- In lines 51-56 we setup the solver for **p_rghFinal**. This correspond to the last iteration in the loop (we can use a tighter convergence criteria to get more accuracy without increasing the computational cost)
- In lines 58-72 we setup the solvers for **U** and **UFInal**.
- In lines 74-80 we setup the solvers for the turbulent quantities, namely, **k** and **epsilon**.

3D Dam break – Free surface flow

The *fvSolution* dictionary

```
82
83      PIMPLE
84      {
85          momentumPredictor   yes;
86          nOuterCorrectors    1;
87          nCorrectors         3;
88          nNonOrthogonalCorrectors 1;
89      }
90
91      relaxationFactors
92      {
93          fields
94          {
95              ".*" 1;
96          }
97          equations
98          {
99              ".*" 1;
100          }
101      }
102
```

- In lines 83-89 we setup the entries related to the pressure-velocity coupling method used (**PIMPLE** in this case). Setting the keyword **nOuterCorrectors** to 1 is equivalent to running using the **PISO** method.
- To gain more stability we can increase the number of correctors (lines 87-88), however this will increase the computational cost.
- In lines 91-101 we setup the under relaxation factors related to the **PIMPLE** method. By setting the coefficients to one we are not under-relaxing.
- The option **momentumPredictor** (line 85), is recommended for highly convective flows.

3D Dam break – Free surface flow

The **`system`** directory

- In the **`system`** directory you will find the following optional dictionary files:
  - *`decomposeParDict`*
  - *`setFieldsDict`*

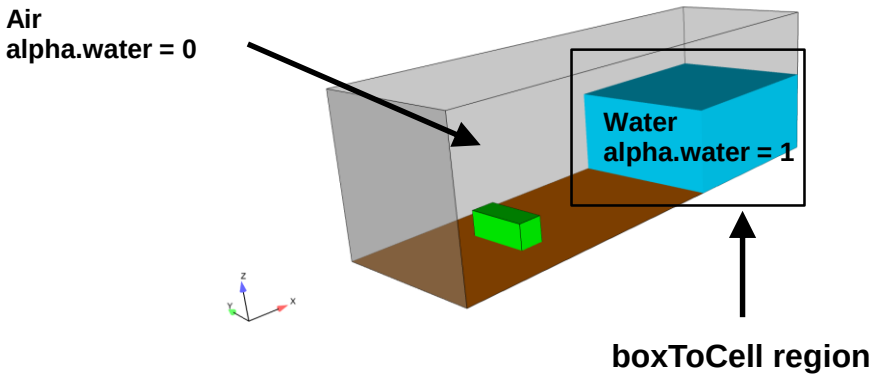
- *`decomposeParDict`* is read by the utility `decomposePar`. This dictionary file contains information related to the mesh partitioning. This is used when running in parallel.
- *`setFieldsDict`* is read by the utility `setFields`. This utility set values on selected cells/faces.

3D Dam break – Free surface flow

The *setFieldsDict* dictionary

```
17  defaultFieldValues
18  (
19      volScalarFieldValue alpha.water 0
20  );
21
22  regions
23  (
24      boxToCell
25      {
26          box (1.992 -10 0) (5 10 0.55);
27          fieldValues
28          (
29              volScalarFieldValue alpha.water 1
30          );
31      }
32  );
```

- This dictionary file is located in the directory **system**.
- In lines 17-20 we set the default value to be 0 in the whole domain (no water).
- In lines 22-32, we initialize a rectangular region (**box**) containing water (**alpha.water 1**).
- In this case, `setFields` will look for the dictionary file *alpha.water* and it will overwrite the original values according to the regions defined in *setFieldsDict*.
- We initialize the water phase because is the primary phase in the dictionary *transportProperties*.
- If you are interested in initializing the vector field **U**, you can proceed as follows **volVectorFieldValue U (0 0 0)**



3D Dam break – Free surface flow

The *decomposeParDict* dictionary

- This dictionary file is located in the directory **system**.
- This dictionary is used to decompose the domain in order to run in parallel.
- The keyword **numberOfSubdomains** (line 17) is used to set the number of cores we want to use in the parallel simulation.
- In this dictionary we also set the decomposition method (line 19).
- Most of the times the scotch method is fine.
- In this case we set the **numberOfSubdomains** to 4, therefore we will run in parallel using 4 cores.

```
17    numberOfSubdomains 4;
18
19    method scotch;
20
```

- When you run in parallel, the solution is saved in the directories **`processorN`**, where **`N`** stands for processor number. In this case you will find the following directories with the decomposed mesh and solution: **`processor0`**, **`processor1`**, **`processor2`**, and **`processor3`**.

3D Dam break – Free surface flow

Running the case

- Let us first generate the mesh.
- To generate the mesh will use snappyHexMesh (sHM), do not worry we will talk about sHM tomorrow.

1. `$> foamCleanTutorials`
2. `$> rm -rf 0`
3. `$> blockMesh`
4. `$> surfaceFeatureExtract`
5. `$> snappyHexMesh -overwrite`
6. `$> createPatch -dict system/createPatchDict.0 -overwrite`
7. `$> createPatch -dict system/createPatchDict.1 -overwrite`
8. `$> checkMesh`
9. `$> paraFoam`

3D Dam break – Free surface flow

Running the case

- Let us run the simulation in parallel using the solver `interFoam`.
- We will talk more about running in parallel tomorrow
- To run the case, type in the terminal:

```
1. $> rm -rf 0
2. $> cp -r 0_org 0
3. $> setFields
4. $> paraFoam
5. $> decomposePar
6. $> mpirun -np 4 interFoam -parallel | tee log.interFoam
7. $> reconstructPar
8. $> paraFoam
```

3D Dam break – Free surface flow

Running the case

- In steps 1-2 we copy the information of the backup directory **`0_org`** into the directory **`0`**. We do this because in the next step the utility `setFields` will overwrite the file *`0/alpha.water`*, so it is a good idea to keep a backup.
- In step 3 we initialize the solution using the utility `setFields`. This utility reads the dictionary *`setFieldsDict`* located in the **`system`** directory.
- In step 4 we visualize the initialization using paraFoam.
- In step 5 we use the utility `decomposePar` to do the domain decomposition needed to run in parallel.
- In step 6 we run the simulation in parallel. Notice that np means number of processors and the value used should be the same number as the one you set in the dictionary *`decomposeParDict`*.
- If you want to run in serial, type in the terminal: `interFoam | tee log`
- In step 7 we reconstruct the parallel solution. This step is only needed if you are running in parallel.
- Finally, in step 8 we visualize the solution.

3D Dam break – Free surface flow

- To plot the sampled data using gnuplot you can proceed as follows. To enter to the gnuplot prompt type in the terminal:

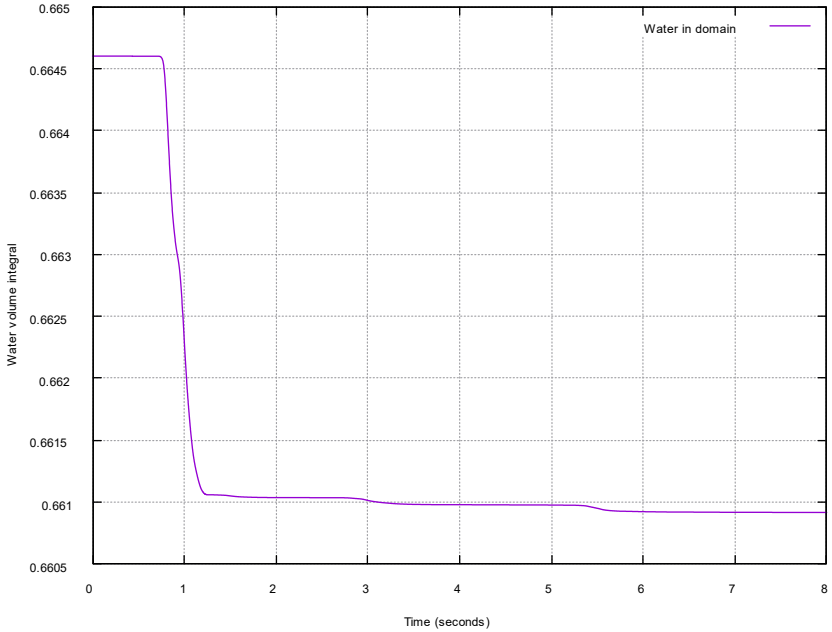
1. ```
   $> gnuplot
   ```

- Now that we are inside the gnuplot prompt, we can type,

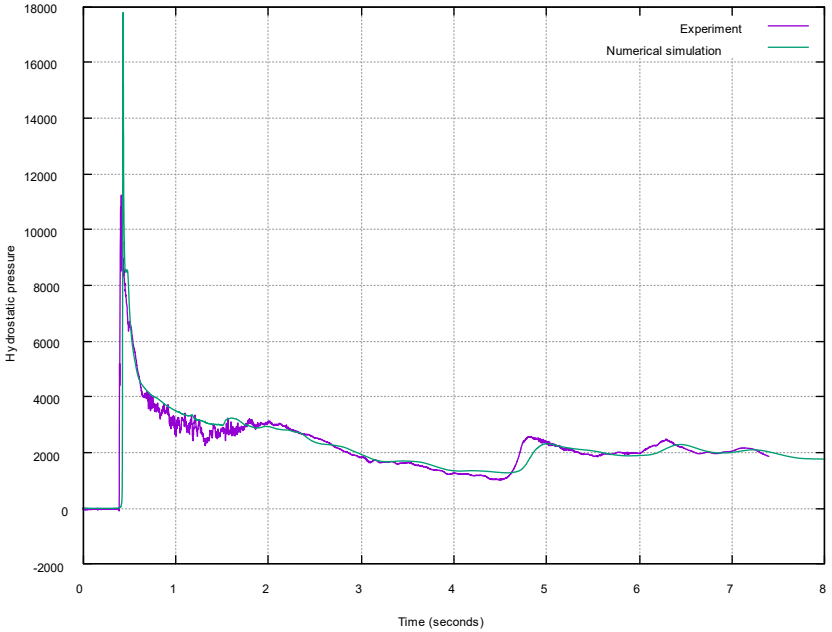
1. ```
   set xlabel 'Time (seconds)'
   ```
2. ```
   set ylabel 'Water volume integral'
   ```
3. ```
   gnuplot> plot 'postProcessing/water_in_domain/0/volRegion.dat' u 1:2 w l title 'Water in domain'
   ```
4. ```
   set xlabel 'Time (seconds)'
   ```
5. ```
   set ylabel 'Pressure'
   ```
6. ```
   plot 'SPHERIC_Test2/case.txt' u 1:2 w l title 'Experiment', 'postProcessing/probes1/0/p' u 1:2 w l title 'Numerical simulation'
   ```
7. ```
   gnuplot> exit
   ```
   To exit gnuplot

3D Dam break – Free surface flow

- The output of steps 3 and 6 is the following:



alpha.water vs. time



p vs. time (at probe 0)

3D Dam break – Free surface flow

The output screen

```
Courant Number mean: 0.0099001831 max: 0.50908228
Interface Courant Number mean: 0.0012838336 max: 0.05362054
deltaT = 0.00061195165
Time = 0.41265658

PIMPLE: iteration 1
smoothSolver:  Solving for alpha.water, Initial residual = 0.00035163885, Final residual = 9.3476388e-11, No Iterations 2
Phase-1 volume fraction = 0.20706923  Min(alpha.water) = -9.1300674e-12  Max(alpha.water) = 1.0000113
MULES: Correcting alpha.water
MULES: Correcting alpha.water
MULES: Correcting alpha.water
Phase-1 volume fraction = 0.20706923  Min(alpha.water) = -1.2354076e-07  Max(alpha.water) = 1.0000113
DILUPBiCGStab:  Solving for Ux, Initial residual = 0.00057936556, Final residual = 2.3207684e-09, No Iterations 1
DILUPBiCGStab:  Solving for Uy, Initial residual = 0.0021990412, Final residual = 7.228845e-09, No Iterations 1
DILUPBiCGStab:  Solving for Uz, Initial residual = 0.00041048425, Final residual = 3.946807e-10, No Iterations 1
DICPCG:  Solving for p_rgh, Initial residual = 0.0013260985, Final residual = 1.2556023e-05, No Iterations 4
DICPCG:  Solving for p_rgh, Initial residual = 1.4873252e-05, Final residual = 8.7706547e-07, No Iterations 13
time step continuity errors : sum local = 2.166836e-08, global = -4.8300033e-11, cumulative = -5.8278026e-05
DICPCG:  Solving for p_rgh, Initial residual = 1.6925332e-05, Final residual = 8.9811533e-07, No Iterations 9
DICPCG:  Solving for p_rgh, Initial residual = 1.1731393e-06, Final residual = 4.991128e-07, No Iterations 1
time step continuity errors : sum local = 1.2328745e-08, global = -3.6165262e-09, cumulative = -5.8281643e-05
DICPCG:  Solving for p_rgh, Initial residual = 8.2834963e-07, Final residual = 4.6047958e-07, No Iterations 1
DICPCG:  Solving for p_rgh, Initial residual = 4.6053278e-07, Final residual = 4.65519e-07, No Iterations 1
time step continuity errors : sum local = 1.1498949e-08, global = -3.1908629e-09, cumulative = -5.8284834e-05
DILUPBiCGStab:  Solving for epsilon, Initial residual = 0.001169828, Final residual = 9.2601488e-11, No Iterations 2
DILUPBiCGStab:  Solving for k, Initial residual = 0.0014561556, Final residual = 9.4651262e-11, No Iterations 2
ExecutionTime = 23.21 s  ClockTime = 24 s

fieldMinMax minmaxdomain write:
    min(p) = -9.8942827 in cell 5509 at location (2.490155 0.025000016 1) on processor 2
    max(p) = 4703.3656 in cell 1485 at location (3.1948336 -0.425 0) on processor 2
    min(p_rgh) = -7.9025882 in cell 1241 at location (0.82088765 -0.20846334 0.043756428) on processor 1
    max(p_rgh) = 4831.247 in cell 3285 at location (3.1948341 -0.475 0.42499986) on processor 2
    min(U) = (-0.96505264 -0.019641482 -0.052664083) in cell 2 at location (2.1879167 -0.42500042 0.024999822) on processor 2
    max(U) = (0.32541708 0.29383224 2.7117589) in cell 5246 at location (0.8884354 0.087713417 0.16296979) on processor 1
    min(alpha.water) = -1.2354076e-07 in cell 2653 at location (0.84202094 -0.10628417 0.0062556498) on processor 1
    max(alpha.water) = 1.0000113 in cell 224 at location (2.6411358 -0.42500003 0.074999874) on processor 2
    min(k) = 0.0041733636 in cell 2510 at location (0.65789113 -0.0062500875 0.0062360099) on processor 1
    max(k) = 0.83402261 in cell 6589 at location (1.2803306 -0.025028634 0.17499623) on processor 1
    min(epsilon) = 0.018352121 in cell 2510 at location (0.65789113 -0.0062500875 0.0062360099) on processor 1
    max(epsilon) = 11.712212 in cell 1933 at location (0.83147515 -0.19630576 0.068753535) on processor 1

volFieldValue water_in_domain write:
    volIntegrate() of alpha.water = 0.66459985
```

Flow courant number

Interface courant number. When solving multiphase flows, is always desirable to keep the interface courant number less than 1.

alpha.water residuals

nAlphaCorr 3

nAlphaSubCycles 1
Only one loop

3 pressure correctors and no non-orthogonal corrections

Tighter tolerance (p_rghFinal) is only applied to this iteration (the final one)

Turbulence variables residuals

Minimum and maximum values of field variables

Volume integral functionObject

3D Dam break – Free surface flow

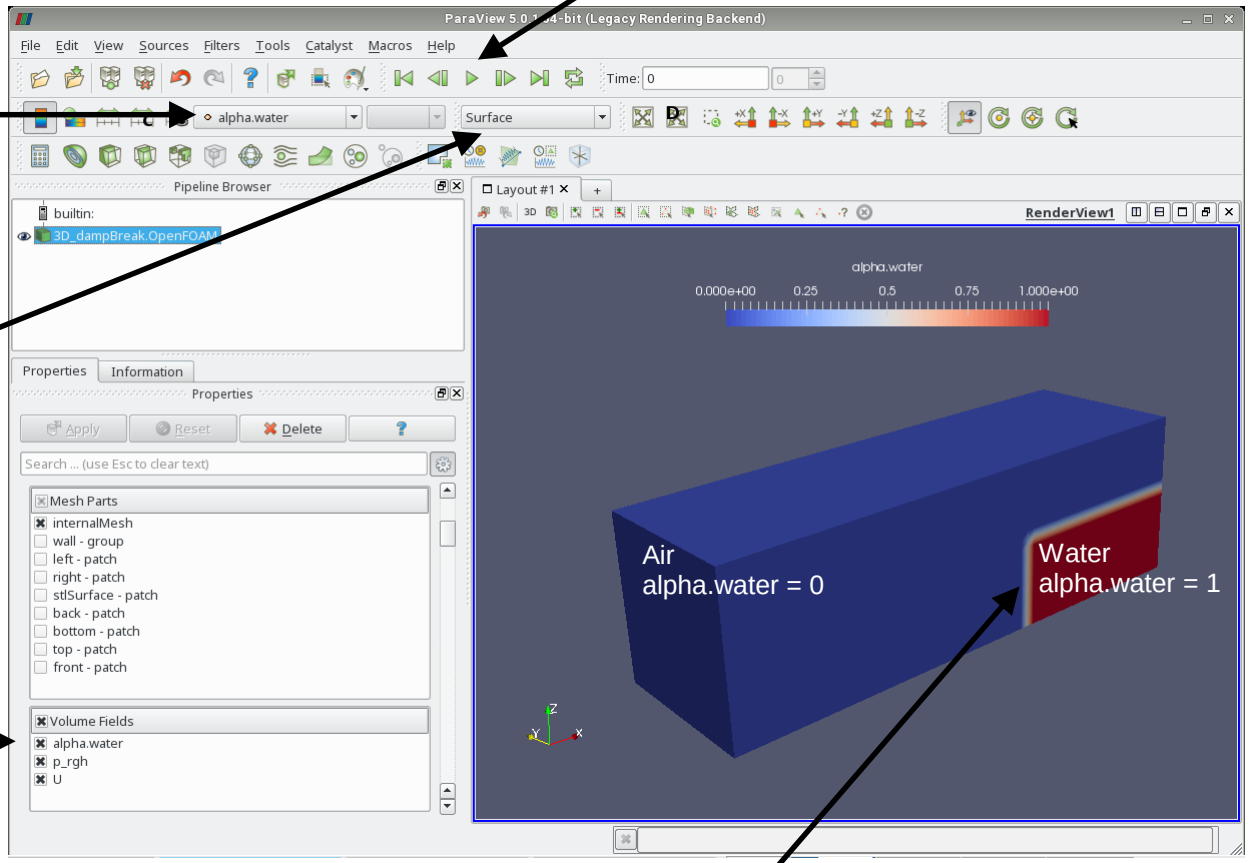
Post-processing multiphase flows in paraFoam

- To visualize the volume fraction, proceed as follows,

4. To animate the solution, press `Play` in the VCR Controls

2. Select **alpha.water** in the Active Variable drop-down menu

3. Select `Surface` in the Representation drop-down menu



1. In the Properties tab select **alpha.water** in Volume Fields

Interface
alpha.water = 0.5

3D Dam break – Free surface flow

Post-processing multiphase flows in paraFoam

- To visualize a surface representing the interface, proceed as follows,

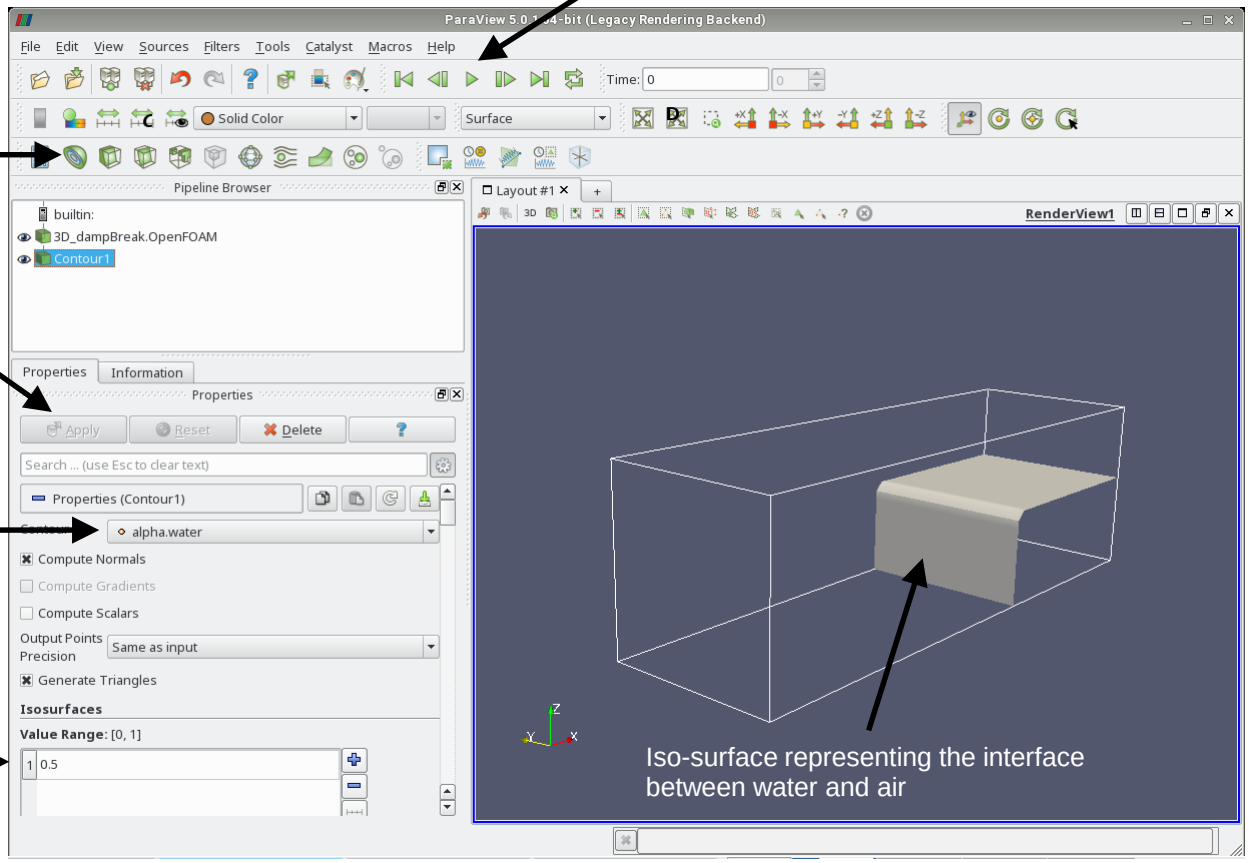
5. To animate the solution, press `Play` in the VCR Controls

1. Select the filter `Contour`

4. Press apply

2. Select **alpha.water** or the field you want to use to plot the iso-surface (it has to be a scalar)

3. Enter the value 0.5 which corresponds to the interface between water and air



3D Dam break – Free surface flow

Post-processing multiphase flows in paraFoam

- To visualize all the cells representing the water fraction, proceed as follows,

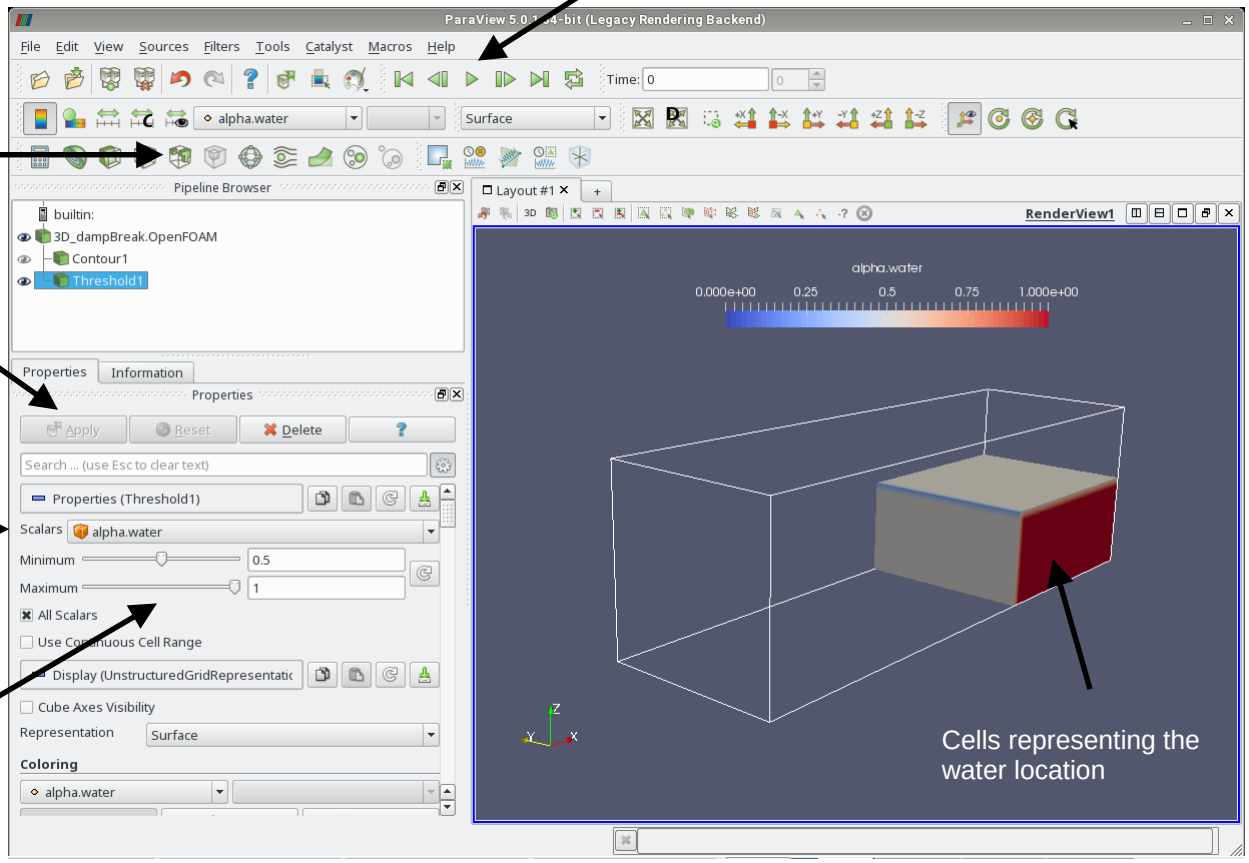
5. To animate the solution, press `Play` in the VCR Controls

1. Select the filter `Threshold`

4. Press apply

2. Select **alpha.water** or the field you want to use to visualize the cells (it has to be a scalar)

3. Select the range you want to visualize. To visualize the water select `Minimum` 0.5 and `Maximum` 1.



3D Dam break – Free surface flow

Exercises

- Instead of using the boundary condition **totalPressure** and **pressureInletOutletVelocity** for the patch **top**, try to use **zeroGradient**. Do you get the same results? Any comments?
  (Hint: this combination of boundary conditions will give you an error, read carefully the screen, you will need to add a fix in the file `fvSolution`**)**
- Instead of using the boundary condition **fixedFluxPressure** for the walls, try to use **zeroGradient**. Do you get the same results? Any comments?
- Run the simulation in a close domain. Does the volume integral of **alpha.water** remains the same? Why the value is not constant when the domain is open?
- Use a **functionObject** to measure the average pressure at the obstacle.
- How many initialization methods are there available in the dictionary `setFieldsDict`?
  (Hint: use the banana method)
- Run the simulation using **Gauss upwind** instead of **Gauss vanLeer** for the term **div(phi,alpha)** (`fvSchemes`). Do you get the same quantitative results?
- Run a numerical experiment for **cAlpha** equal to **0**, **1**, and **2**. Do you see any difference in the solution? What about computing time?
- Use the solver **GAMG** instead of using the solver **PCG** for the variable **p_rgh**. Do you see any difference on the solution or computing time?
- Increase the number of **nOuterCorrector** to 2 and study the output screen. What difference do you see?
- Turn off the MULES corrector (**MULESCorr**). Do you see any difference on the solution or computing time?
- If you set the gravity vector to (0 0 0), what do you think will happen?
- Try to break the solver and identify the cause of the error. You are free to try any kind of setup.