Final project: simulation of a bubble column

Group 2:
Gerasimos Chourdakis, Vasileios Magioglou

July 27, 2016

Contents

1 Introduction 2

2 Physics, numerics and methodology 2
2.1 Numerical formulations of bubbleFoam 2
2.2 The bubbleFoam solver 3
2.3 Alternative solvers 5

3 Setup 7
3.1 Starting point 7
3.2 Mesh setup and refinement 7
3.3 Initial and boundary conditions 11
3.4 Constants and solver configuration 13
3.5 Parallel execution 14

4 Results and discussion 14
4.1 Main case 14
4.2 Bubble diameter exploration 15
4.3 Inlet velocity exploration 20
4.4 Interaction with obstacles 20
4.5 Solver aspects 22

5 Conclusion 27
5.1 Lessons learned 27
5.2 Future work 27

1 Introduction

In this project we simulated the flow inside a 3D, cylindrical bubble column using the application `bubbleFoam` of OpenFOAM. We studied different values for the diameter of the bubbles and the inlet velocity. Next, we simulated the interaction with an obstacle and we also studied some particular aspects and problems that we faced during our experiments. In this report, a quick introduction to the model is provided, as well as an extensive presentation of the setup of our case. This gave us a better understanding of the underlying phenomena and directions for further studying are presented at the end.

Please not that, apart from the results shown here, videos and demo cases can also be found in https://tinyurl.com/gl4fhzo (hosted in our Dropbox).

2 Physics, numerics and methodology

2.1 Numerical formulations of bubbleFoam

The numerical solution of the two-phase equations relies on a segregated algorithm based on the PISO procedure extended to two-phase flows. The momentum equations are manipulated to stabilize the system of equation at the limits of the range of volume fractions, to avoid singularities. The details of the numerical methodology are briefly summarized below in the subsections.

2.1.1 Phase momentum equation

The momentum equation is rewritten in non-conservative form to extract the volume fraction from the transport terms. Renaming the total stress tensor, decomposing it in a diffusive component and a correction term and also introducing the total phase velocity, the equation becomes:

$$\begin{aligned}\frac{\partial U_{\mathrm{a}}}{\partial t} + U_{\mathrm{a}}^T \cdot \nabla U_{\mathrm{a}} - \nabla \cdot (v_{\mathrm{a,eff}} \nabla U_\phi) + \nabla \cdot R^C_{\phi,\mathrm{eff}} + \frac{\nabla \alpha_a}{\alpha_a} \cdot R^C_{\mathrm{a,eff}} = \\ = -\frac{\nabla p}{\rho_a} + g + \frac{\alpha_{\mathrm{b}}}{\rho_{\mathrm{a}}}[A_d(U_b - U_a) + A_l + A_v m(\frac{dU_b}{dt}|_b - \frac{dU_a}{dt}|_a)]\end{aligned} \tag{1}$$

The discretized equations can be represented as

$$U_a = \frac{H_a}{A_a} - \frac{\nabla p}{\rho_a A_a} \tag{2}$$

$$U_b = \frac{H_b}{A_b} - \frac{\nabla p}{\rho_b A_b} \tag{3}$$

These equations will be used to correct the velocity after the pressure field is updated.

2.1.2 Phase continuity equation

The phase continuity equations have to be solved ensuring that the phase fraction of each phase is kept between zero and one. To obtain this result, the dispersed phase continuity equation is rewritten as a function of the mean and relative velocity. Rewriting the phase velocity in terms of the relative velocity and the mean velocity weighted on the phase fractions we find

$$U_a = U + a_b U_r \tag{4}$$

Substituting into the phase continuity equation, a new expression is obtained

$$\frac{\partial \alpha_a}{\partial t} + \nabla \cdot (\alpha_a U) + \nabla \cdot (\alpha_a (1 - \alpha_a) U_r) = 0 \tag{5}$$

which, if iteratively (Note that the equation is not linear) solved in a fully implicit manner, provides a bounded solution for the phase fraction field.

2.1.3 Pressure equation

The pressure equation is obtained imposing that the divergence of the mixture flux is zero

$$\nabla \cdot \phi = \nabla \cdot (\alpha_{\mathrm{a,f}} \phi_a + \alpha_{\mathrm{b,f}} \phi_b) = 0 \tag{6}$$

The phase fluxes are obtained by

$$\phi_a = \phi_a^* - \frac{1}{\rho_a A_a}|_f S_f \nabla p + \phi_{\mathrm{dg,a}} \tag{7}$$

$$\phi_b = \phi_b^* - \frac{1}{\rho_b A_b}|_f S_f \nabla p + \phi_{\mathrm{dg,b}} \tag{8}$$

2.2 The bubbleFoam solver

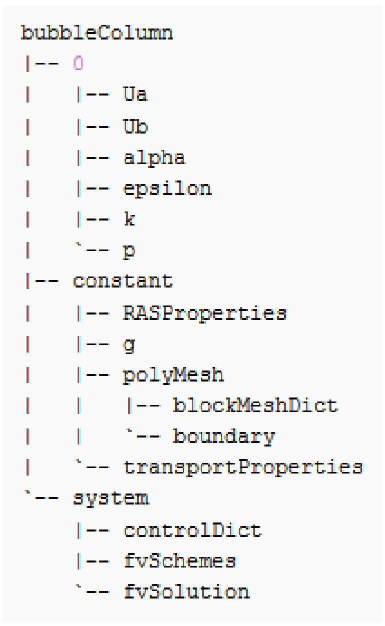
The solver bubbleFoam in OpenFOAM was selected to simulate the bubble column. This is based on theoretical study and testing of other solvers, e.g. twoPhaseEulerFoam. This section is largely drawn from openfoanwiki.net. The bubbleFoam solver is a two-phase solver based on the Euler-Euler two-fluid methodology, suitable to compute dispersed gas-liquid and liquid-liquid flows. In the Euler-Euler two-fluid approach, the phases are treated as interpenetrating continua, which are capable of exchanging properties, like momentum, energy and mass one with the other. Typical applications of the two-fluid approach, as implemented in bubbleFoam, are bubble columns, stirred tank reactors, static mixers etc.

The solution procedure adopted in the bubbleFoam solver is summed up as follows:

1. Solve the phase continuity equations

2. Update lift, drag and virtual mass coefficients
3. Construct the momentum equation matrix
4. Predict the phase velocity fields, without considering the pressure gradient at this stage
5. Solve the pressure equation
6. Correct the velocities with the new pressure field, and update the phase fractions
7. Solve the transport equations for the turbulence quantities

The structure of a typical bubbleFoam case is represented in the following directory tree shown in figure 1, which lists the directories and the files contained in the bubbleColumn tutorial.



```
bubbleColumn
|-- 0
|   |-- Ua
|   |-- Ub
|   |-- alpha
|   |-- epsilon
|   |-- k
|   `-- p
|-- constant
|   |-- RASProperties
|   |-- g
|   |-- polyMesh
|   |   |-- blockMeshDict
|   |   `-- boundary
|   `-- transportProperties
`-- system
    |-- controlDict
    |-- fvSchemes
    `-- fvSolution
```

Figure 1: bubbleFoam case setup and structure

2.2.1 bubbleFoam capabilities

The bubbleFoam solver implements the equations that are presented in the following, for the simulation of gas-liquid flows. The model undergoes the following assumptions:

- Phases are incompressible
- The dispersed phase particle diameter is constant

- The flow is isothermal
- Only momentum exchange is accounted for in the momentum transport equations

The main features of the solver are the following:

- Capability to solve for dispersed two-phase flows with strong density ratio
- Robust solution algorithm, able to deal with complete flow separation
- Turbulence modelling through $k-\epsilon$ model and standard wall functions

2.2.2 bubbleFoam limitations

The bubbleFoam solver currently has the following limitations:

- Only one dispersed phase and a continuous phase can be described. It is not possible to account for multiple dispersed phases (i.e. represent a dispersed phase diameter distribution)
- The diameter of the particles1 constituting the dispersed phase is assumed to be constant. Aggragation, breakage and coalescence phenomena are not accounted for
- The drag coefficient is computed as a blend of the drag coefficients evaluated for each phase on the basis of the phase fractions, and no alternative drag models are available
- The interaction between the phases happens only through the momentum exchange term in the corresponding momentum equations:
  - It is not possible to model the heat transfer between the phases
  - It is not possible to model the mass transfer between the phases
  - No chemical reaction model is available

2.3 Alternative solvers

There are alternative solvers to be used for the simulation of a bubble column. Some of these are the twoPhaseEulerFoam, which we implemented as an additional test case and the interFoam provided by OpenFOAM.

2.3.1 twoPhaseEulerFoam solver

Two incompressible fluid phases with one phase dispersed are solved using this solver. Both the phases are described using the Eulerian conservation equations and thus it is referred as Euler-Euler model. Each of the phases are treated as a continuum in this approach. This solver is considered an extension of the bubbleFoam solver. It offers the option of particle-particle interactions, as well as more drag models to choose from to estimate the drag coefficients for each phase. In comparison to the bubbleFoam solver structure, it includes three additional files in the constant folder which help in setting different models and properties for simulating two phase flows (InterfacialProperties, kineticTheoryProperties, ppProperties). There is also an extra boundary condition in the 0 folder, called Theta and represents the granular temperature.

2.3.2 interFoam solver

According to "OpenFOAM User Guide" interFoam is a solver for 2 incompressible fluids, which tracks the interface and includes the option of mesh motion. It uses the multidimensional universal limiter for explicit solution (MULES) method, created by OpenCFD, to maintain boundedness of the phase fraction independent of underlying numerical scheme, mesh structure, etc. The choice of schemes for convection are therefore not restricted to those that are strongly stable or bounded, e.g. upwind differencing. We have not implemented interFoam for bubbleColumns, but still remains an option for this type of simulations.

2.3.3 MPPICInterFoam solver

The new MPPICInterFoam solver combines Multi-Phase Particle In Cell (MP-PIC) Lagrangian cloud modelling for particles with high volume loading, with two phase isothermal and immiscible fluids modelling using Volume of Fluid (VOF) interface capturing. Particles are modelled using the same approach adopted in the MPPICfoam solver, including the effect of the volume fraction of particles on the continuous phase. To limit the particles crossing the interface, typically from liquid to gas, a new interface force was introduced. This force acts in the direction normal to the interface and only near the interface according to the expression

$$F_{\text{interface}} = Cm\nabla(a_w) \tag{9}$$

where m is the particle mass, a_w the liquid volume fraction, and C a model coefficient. These are specified in the particleForces sub-dictionary of the cloudProperties file. The solver includes support for Multiple Reference Frame (MRF) forces, specified via the MRFProperties file, and other forces via fvOptions. Turbulence modelling is generic, supporting RAS, LES/DES or laminar models.

We discovered this solver recently after the completion of our project. It was released in June 2016. It seems like a promising improved solver to simulate two phases in particular bubble columns, while at the same time it offers enhanced visualization effects for the discrete separation of the phases.

3 Setup

3.1 Starting point

We designed our case based on the *bubbleColumn* tutorial of the `bubbleFoam` solver, of OpenFOAM version 2.0.1. An example of the results of this tutorial is shown in figure 2. This is a 2D case that uses a uniform grid and sets a continuous air inlet from the whole area of the bottom boundary. We wouldn't expect to observe individual bubbles but, eitherways, these rough results don't help us much to understand the phenomena that take place.

We extended this case to simulate the introduction of the air only through a small hole at the bottom and with pulses. This gave us structures that behaved like bigger bubbles. We will refer to these simply as "bubbles" in the rest of the text, except if noted differently. We also transferred the problem from 2D to 3D, creating a coarse cylindrical mesh and refining in the most active areas. Furthermore, we introduced obstacles and explored different inlet scenarios.

3.2 Mesh setup and refinement

In our case we used a cylindrical mesh, created with `blockMesh` and partially refined with `refineMesh`. The full-size mesh ($h = 0.5\,\mathrm{m}$) for our main case is shown in figure 3 and a fraction of the source `blockMeshDict` is shown in listing 1. The mesh is constructed by five blocks, i.e. one central pillar and four peripheral blocks. The edges of the blocks are of type `arc`. All the blocks have the same number of cells, but the central pillar is smaller and therefore denser.

The mesh is then partially refined using the tools `setSet` and `refineMesh`. Two areas are of specific interest: the central pillar in which the bubbles are expected to travel and the volume around the interface with the atmosphere. These are defined in the file `refine.setSet`, as shown in listing 2. The refinement in done one time (splitting to two) towards all the three directions, as set in the `refineMeshDict` (not listed here).

Listing 1: Fraction of the `blockMeshDict` used in the main case.

```
// 16 vertices (8 per z): on each z-face,
// an outer and an inner square
```

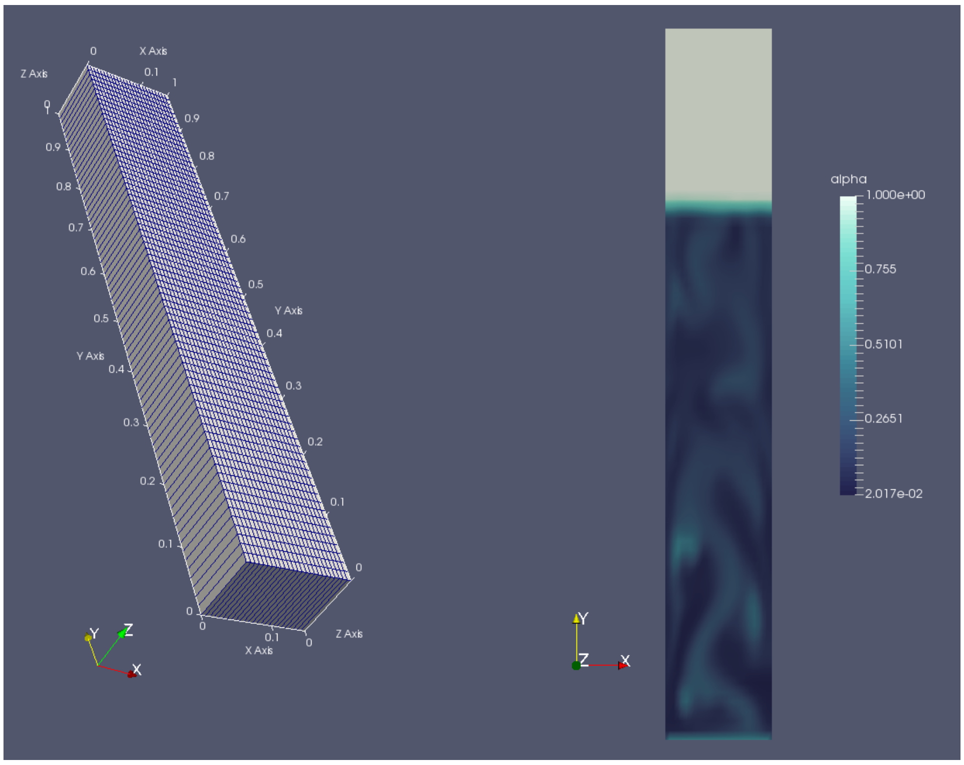


Figure 2: *bubbleColumn* tutorial of the solver `bubbleFoam`, used as a base for our project. A pseudo-2D uniform grid is used, with continuous air inlet from the whole bottom boundary. The size of bubbles introduced is $d_\mathrm{a} = 3\,\mathrm{mm}$ and the inlet velocity $U_{\mathrm{a},y} = 0.1\,\mathrm{m/s}$.

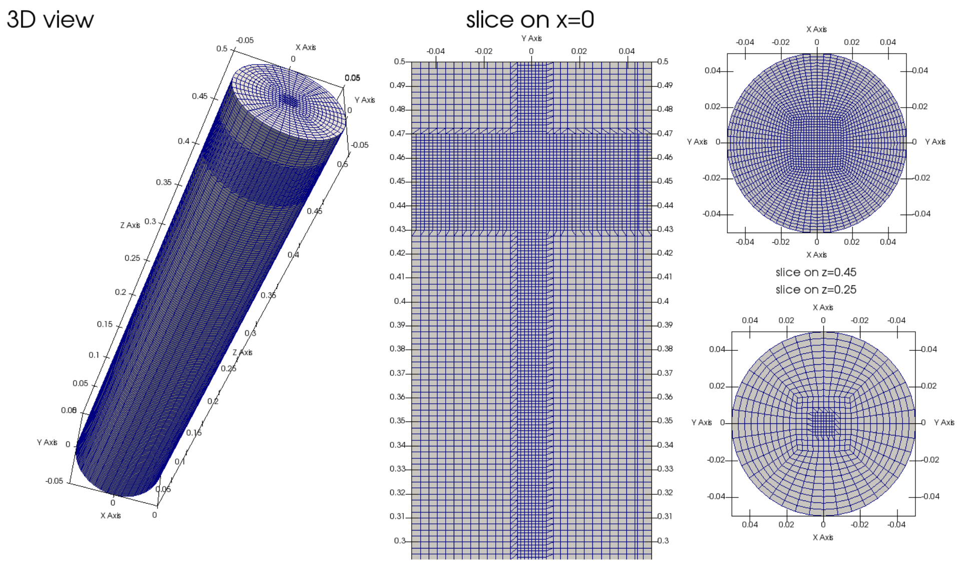


Figure 3: Mesh for the main case: 3D cylinder of 0.50 m length and 0.10 m diameter. The base mesh (without refinement) consists of 200 cells in the z direction and 30 cells towards the x and y directions. Two zones are refined once (the cell size is halved): a symmetric rectangular central pillar with width 0.01 m and the volume of the cylinder between $z = 0.43$ m and $z = 0.47$ m. These parts correspond to the expected "bubble" lift volume and a zone around the interface with the atmosphere. In total, the mesh consists of 228 360 cells. The flow direction is set towards the positive z.

```
vertices
(
    (-3.535534e+01 -3.535534e+01 0.000000e+00) // 0
    ...
    (-1.414214e+01  1.414214e+01 0.5000e+03) // 15
);

// 5 blocks: a central pillar and 4 peripheral
blocks
(
hex (0 1 5 4 8 9 13 12)   pipe (10 10 200)
  edgeGrading (1.000000 ...[11 times])
...
hex (4 5 6 7 12 13 14 15) pipe (10 10 200)
  edgeGrading (1.000000 ...[11 times])
);

// 16 arc-time edges for the inner and outer "circles"
edges
(
arc 0 1 (-9.184851e-15 -5.000000e+01 0.000000e+00)
...
arc 15 12 (-1.500000e+01 1.836970e-15  0.5000e+03)
);

patches
(
patch wall(...)
patch bottom(...)
patch atmosphere(...)
);
```

Listing 2: `refine.setSet` file to select the cells to be refined.

```
# Central pilar
cellSet toBeRefined new boxToCell
  ( -0.005 -0.005 0.0 ) ( 0.005 0.005 0.5 )

# Interface with the atmosphere
cellSet toBeRefined add boxToCell
  ( -0.05 -0.05 0.40 ) ( 0.05 0.05 0.5 )
```

In the case of the obstacle (figures 13 and 14), another iteration of refinement is done for the area around the introduced obstacle (before it gets introduced), as set in a similar `refineSphere.setSet` file. The volume of the obstacle is then selected again with the `setSet` tool, using the `obstacle.setSet` file shown in listing 3 and removed using the `subsetMesh` tool to overwrite the initial mesh. An alternative approach would have been to use the `snappyHexMesh`, that would also change the shape of the cells around the object. We chose this method as it was simpler since we were already working with `blockMesh`. The exact way of calling these tools can be found in the `Allrun`, `Allparrun`, `Allparrun_job` of the demo cases.

Listing 3: `obstacle.setSet` file to select the cells to be removed (obstacle).

```
cellSet wallObstacle new
cellSet wallObstacle invert

cellSet wallObstacle delete sphereToCell
  (0.0 0.0 0.05) 0.005
```

3.3 Initial and boundary conditions

In the `blockMeshDict` (listing 1) the following patches are defined: `wall`, i.e. the cylinder that encapsulates the domain (and the sphere, in the case of the obstacle), `bottom`, i.e. the bottom wall (the same as `wall`, but can be treated differently if desired to do so) and `atmosphere`, i.e. the top boundary that connects to the outer world. An additional patch is created for the inlet, named `bubble`, using the `setSet` and `createPatch` tools. The configuration file is shown in listing 4. The `createPatchDict` is not listed here.

The initial and boundary conditions are shown in table 1. The `alpha` is initialized to be 0 (continuous phase) at the bottom of the column and 1 (dispersed phase) at the top of the column. This is done with the `setFields` tool, using the `setFieldsDict` shown in listing 5. At the inlet, it is set to be either 0 or 0.95, using a `groovyBC` with the condition `mag(Ua) > 0`. We observed in our early experiments that a non-pure inlet stream was helping the stability of the solution. On the walls and on the top boundary the `alpha` is simply set to not change.

The velocities `Ua` and `Ub` are initialized to zero and no-slip walls are assumed. For the atmosphere, an `inletOutlet` boundary condition is used in both cases. At the inlet the water velocity is always set to zero, while the air velocity varies with time. For this, the `timeVaryingUniformFixedValue` boundary condition is used, which requires the OpenFOAM version 2.0.1 (it doesn't work with the v.2.1.1 that is the default in the computer lab). The file `Ua_time.dat` provides the pattern, as shown in listing 6, that is repeated.

The pressure `p` is initialized to 10^{-5} and the same value is set for the atmosphere (outlet), as required for compressible flows. On the walls, the gradient is set to zero using `buoyantPressure`. On the inlet, `zeroGradient`

Field	Scope	Type/Value	Notes
gas fraction	initial	0 or 1	`setFields`
(`alpha`)	bubble	0 or 0.95	`groovyBC`
	atmosphere	`zeroGradient`	-
	wall	`zeroGradient`	-
air velocity	initial	(0 0 0)	-
(`Ua`)	bubble	0 or (0 0 0.2)	time varied
	atmosphere	`inletOutlet` 0 0	(0 0 0)
	wall	(0 0 0)	-
water velocity	initial	(0 0 0)	-
(`Ub`)	bubble	(0 0 0)	-
	atmosphere	`inletOutlet` 0 0	(0 0 0)
	wall	(0 0 0)	-
pressure	initial	10^{-5}	-
(`p`)	bubble	`zeroGradient`	-
	atmosphere	10^{-5}	-
	wall	`buoyantPressure` 0	-

Table 1: Initial and boundary conditions for the main case.

is used. The former corresponds to the static pressure and was used in the tutorial. The latter was chosen by us and we didn't face any problems.

Listing 4: `makeFaceSet.setSet` file to select the inlet cells.

```
faceSet fbubble new boxToFace
  (-0.002 -0.002 -0.0001) (0.002 0.002 0.0001)
```

Listing 5: `setFieldsDict` file to partially fill the column.

```
defaultFieldValues(
    volScalarFieldValue alpha 1
);

regions(
    boxToCell {
        box (-0.05 -0.05 0) (0.05 0.05 0.45);
        fieldValues(volScalarFieldValue alpha 0);
    }
);
```

Listing 6: Time pattern of the `Ua` at the inlet

```
(
  (0.000 (0 0 0.2))
  (0.020 (0 0 0.2))
  (0.021 (0 0 0.0))
  (0.080 (0 0 0.0))
  (0.081 (0 0 0.2))
)
```

3.4 Constants and solver configuration

The `bubbleFoam` solver application requires several parameters to be set for the two phases in the `transportProperties` file, as discussed in section 2.2. We used the same values that we found in the tutorial and we experimented with the values for `da` (as well as the inlet velocity and frequency). We tried three different diameters: 1 mm, 3 mm (main case), 9 mm. The values for our main case are shown in listing 7.

Listing 7: The `transportProperties` file for our main case.

```
rhoa      rhoa [ 1 -3 0 0 0 0 0 ] 1;
rhob      rhob [ 1 -3 0 0 0 0 0 ] 1000;

nua       nua [ 0 2 -1 0 0 0 0 ] 1.6e-05;
nub       nub [ 0 2 -1 0 0 0 0 ] 1e-06;

da        da [ 0 1 0 0 0 0 0 ] 0.003;
db        db [ 0 1 0 0 0 0 0 ] 0.0001;

Cvm       Cvm [ 0 0 0 0 0 0 0 ] 0.5;
Cl        Cl [ 0 0 0 0 0 0 0 ] 0;
Ct        Ct [ 0 0 0 0 0 0 0 ] 1;
```

We set the flow to laminar, although the solver also supports the $k - \epsilon$ turbulence model. The files for the initial and boundary conditions are also provided as a copy of those found in the tutorial.

In the `controlDict` an adjusted timestep in set, bounded by a Courant number of 0.1. The respective fraction of the file is shown in listing 8. The 3rd-party library `libgroovyBC.so` needs also to be set here.

Listing 8: Fraction of the `controlDict` file.

```
deltaT            1e-4;

adjustTimeStep    yes;

maxCo             0.1;
maxAlphaCo        0.1;

```

```
maxDeltaT           0.05;

libs ( "libOpenFOAM.so" "libgroovyBC.so" );
```

The `fvSchemes` and `fvSolution` files are inherited from the tutorial without any changes. A `DIC` (diagonal incomplete Cholesky)-preconditioned `PCG` (preconditioned conjugate gradient) solver is used for the pressure, while a `DILU` (diagonal incomplete LU)-preconditioned `PBiCG` (preconditioned bi-conjugate gradient) is used for the gas fraction. These files are not listed here but can be found in the demo cases.

3.5 Parallel execution

The main case was consisted of 228 360 cells and approximately 2 s of simulated time were required in order to observe the full evolution of the phenomenon, i.e. several bubbles traveling all the way to the surface. To deal with this, we decomposed the case and solved it with 2 or 4 processes, depending on the system. The simple decomposition defined in the `decomposeParDict` (not listed here) was (2 1 1) or (2 2 2) respectively. Then, the case was ran either locally or on the *legends* system. The delivered demo cases are adjusted for the *buddies* cluster. The case can be solved locally with the `Allparrun` or on the *buddies* with the `Allparrun_job` script.

4 Results and discussion

4.1 Main case

The main results of the main scenario are shown in figures 4, 5, 6 and 7. In figure 4 we can observe that the bubbles make also the continuous phase to move, in a way that it returns deep into the core of the column. This effect also slows down the bubbles, as shown in figure 5. In the `alpha` plots, discrete areas of air ("bubbles") are observed. They get formed round, but eventually their shape is affected by the drag forces. The bubbles that are already downsream seem to also pull the bubbles following. On the top level, we observe that the surface gets higher, as air volume is introduced in the column. The bubbles interact with the surface and change its shape over time.

In figure 5 we can observe again discrete bubbles, as well as their acceleration due to lift and their deceleration, probably due to the recirculation, taking in mind the height that this starts to become important. From the horizontal profiles we can observe that the air stays together in the center of the column and that the center of the bubbles is moving faster than the rest of their volume, something that explains the narrower shape of the bubbles

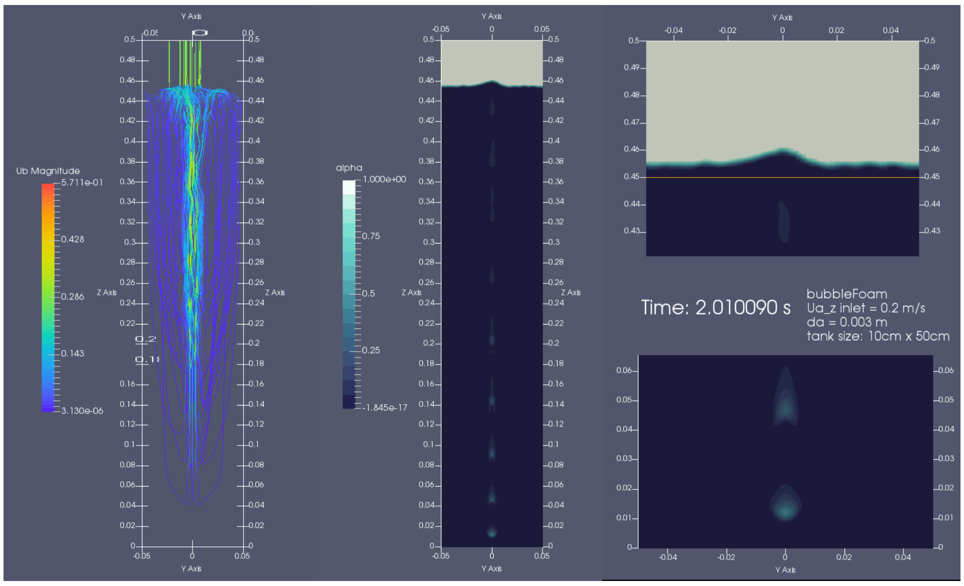


Figure 4: Continuous phase velocity magnitude and streamlines (left), phase distribution in the column (center) and zoom-in to the top and the bottom part of the column (right) after a quasi-steady state has been achieved. The flow of the distributed phase makes also the continuous phase to move up and, after facing the surface, to recirculate towards the bottom. Formation of individual "bubble pulses" is observed, whose size and shape vary as they travel towards the top. The surface level is moved higher than initially set (orange line) and the bubbles interact with the surface. In this scenario, 3 mm air is injected from the bottom with initial velocity of 0.2 m/s towards the z direction, once every 0.08 s and with injection duration of 0.02 s.

near the top. In figure 6 an overview of the pressure and the magnitudes of the individual velocities is shown.

In figure 7 the residuals are shown. The residuals are quite low and generally constant, with short jumps of small magnitude. These jumps are expected, since we are periodically starting and stopping the introduction of air abruptly.

4.2 Bubble diameter exploration

We tested three different values for the dispersed phase diameter, `da`: 1 mm, 3 mm and 9 mm. The results after 1 s of simulated time are shown in figure 8. We can observe that bigger bubbles arrive at the surface earlier and that smaller bubbles tend to mix together and with the continuous phase more, getting dispersed on the horizontal plane. In figure 9, the different bubble velocities are depicted, that explain the earlier observation.

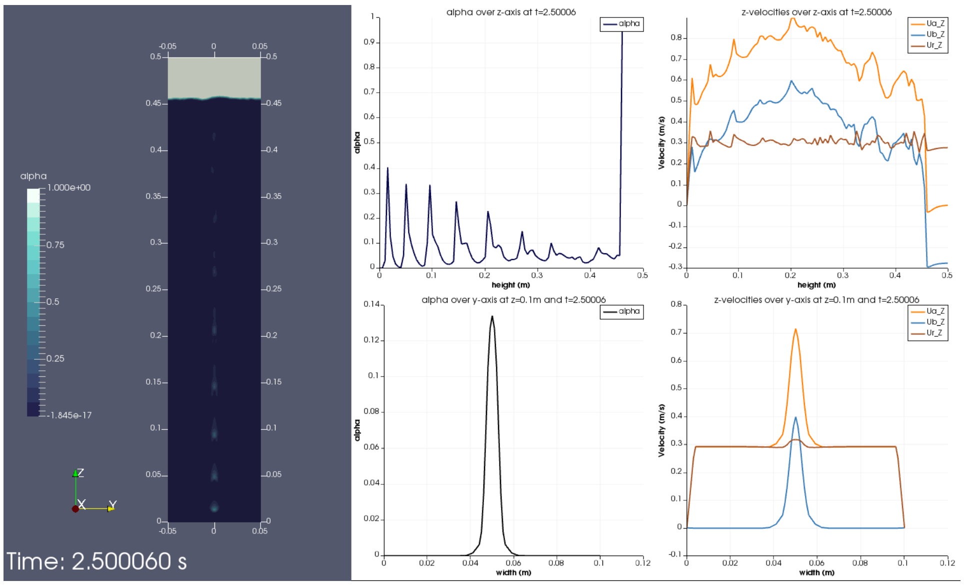


Figure 5: Phase distribution (left) and line plots (center, right) for the main case, at $t = 2.5\,\mathrm{s}$. **Top-center:** `alpha` (gas fraction) distribution across the z-axis. Although the area between the individual dispersed phase "bubbles" is not completely occupied by the continuous phase, there is a clear distinction between these, that becomes less clear near the top. **Top-right:** z-velocities of the dispersed (`Ua`, orange) and of the continuous (`Ub`, blue) phase, as well as relative velocity between the phases (`Ur`, red). In this quasi-steady state, the phases are accelerated up to a maximum near the middle of the column. After this point, where also the recirculating streamlines shown in figure 4 start becoming dense, both phases are decelerating. **Bottom-center:** `alpha` profile over y-axis at $z = 0.1\,\mathrm{m}$ and $t = 2.5\,\mathrm{s}$. The dispersed phase is mainly accumulated in the center axis of the column. **Bottom-right:** z-velocities profiles over the y-axis at $z = 0.1\,\mathrm{m}$ and $t = 2.5\,\mathrm{s}$. The dispersed phase is moving faster than the continuous phase and the center of the "bubble" is moving faster than both the outer part of the "bubble" and the center of the continuous phase.

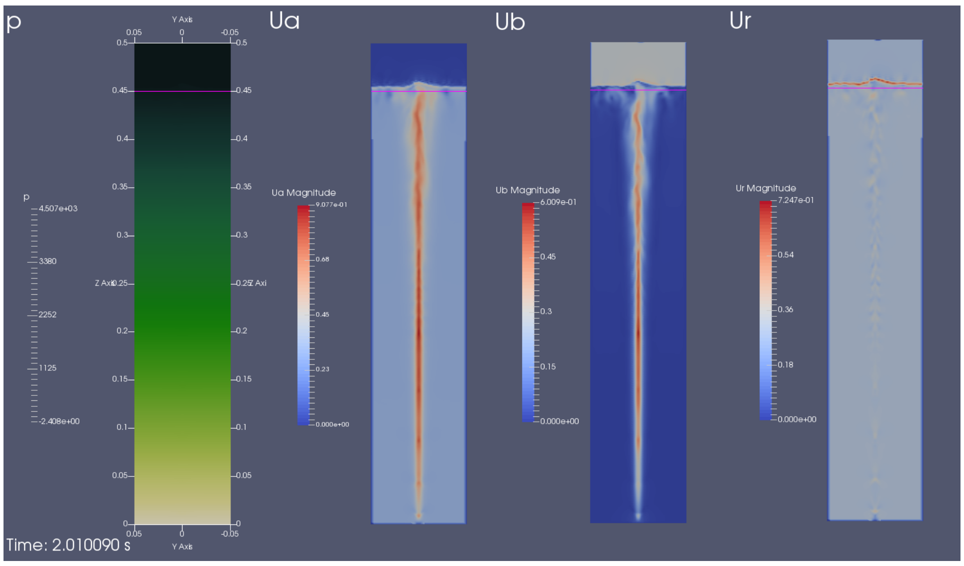


Figure 6: From left to right: pressure, dispersed phase velocity magnitude, continuous phase velocity magnitude and relative velocity magnitude at $t = 2\,\mathrm{s}$ for the main case. The pink line shows the initial surface level. The higher relative velocity in the center of the bubbles, presented in figure 5 is also shown here. Please note that the pressure scale colors go from green (low va)ues lto white (high values), as there was a rendering issue with this.

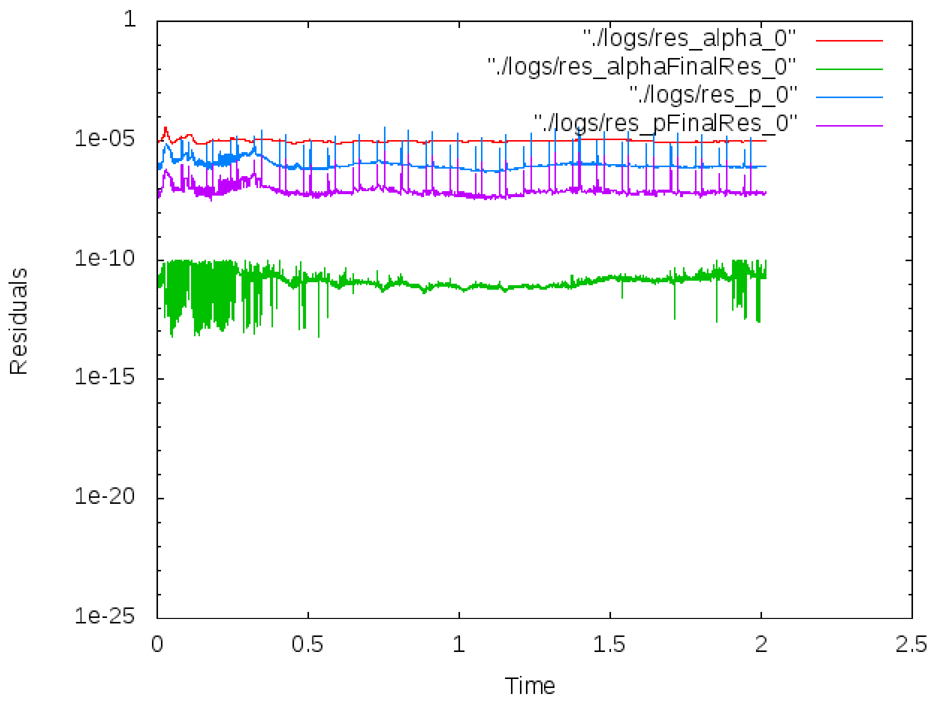


Figure 7: Residuals for the main case. The short extremes are expected, due to the periodic injection of bubbles with steep time gradients.

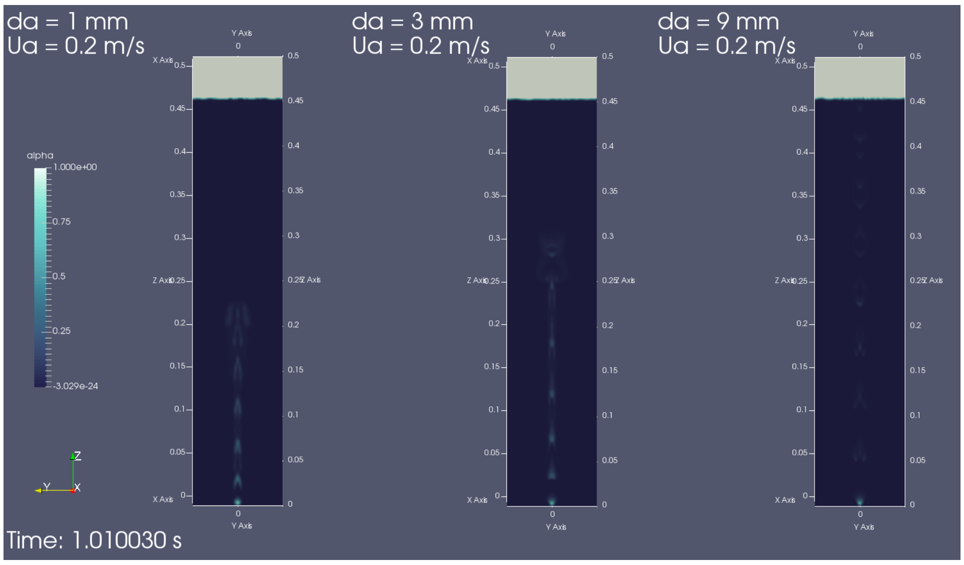


Figure 8: Phase distribution for the main case (center), that sets the diameter of the bubbles to 3 mm and for the same case with $d_a = 1$ mm (left) and $d_a = 9$ mm (right). In the case of smaller bubbles, the distributed phase areas tend to mix, while in the case of bigger bubbles, these areas tend to stay more distinctive. Also, bigger bubbles travel faster, arriving at the surface level earlier. In all cases, the inlet velocity is set to 0.2 m/s and the inlet is a square patch with side 4 mm. Different inlet size or finer grid did not change these observations.

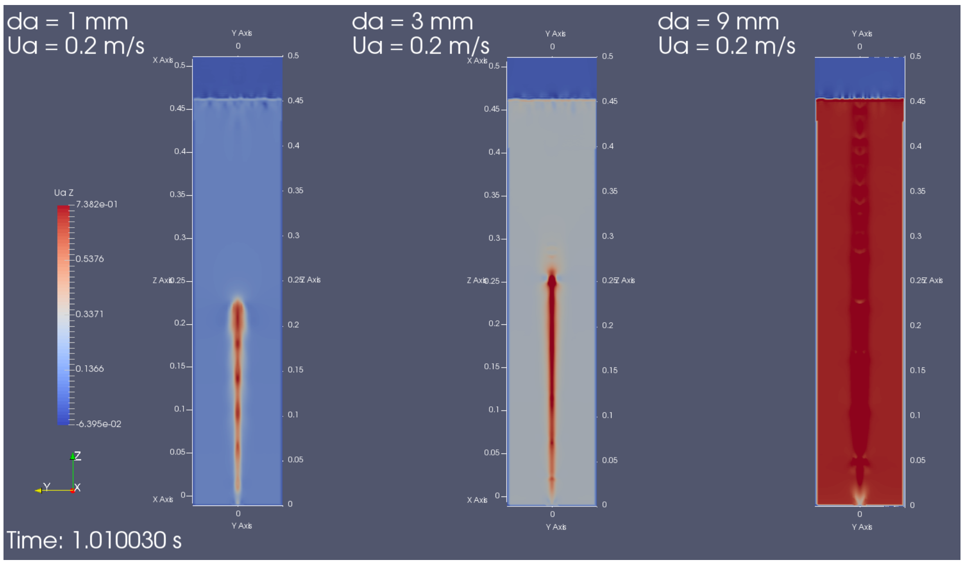


Figure 9: Dispersed phase velocity towards the z-axis for the main case (center), that sets the diameter of the bubbles to 3 mm and for the same case with $d_\mathrm{a} = 1$ mm (left) and $d_\mathrm{a} = 9$ mm (right). The bigger bubbles are traveling with higher velocities in comparison to the smaller. In all cases, the inlet velocity is set to 0.2 m/s.

4.3 Inlet velocity exploration

Apart from different bubble diameters, we also tested different air velocities at the inlet. We tested the values of $U_{\mathrm{a},z} = 0.07$ m/s, 0.2 m/s and 0.6 m/s. The distribution of the bubble areas in the column changes, with lower-speed bubbles tending to stay together, while higher-speed bubbles tend to mix with the continuous phase. We also observe that higher velocity at the inlet gives also higher air velocity inside the column, as well as higher velocity of the continuous phase, as expected. The results are shown in figures 10, 11 and 12.

4.4 Interaction with obstacles

Following the main scenario, we introduced an obstacle to study how the dispersed phase behaves after it. On the one hand, this would give us some insight on the capabilities of the solver. On the other hand, this is a problem with industrial interest, as obstacles are sometimes used to break big bubbles to smaller in bioreactors. The results are shown in the figures 13 and 14.

In figure 13 the geometry and mesh are shown, as well as the separation of big bubble areas to smaller. In figure 14 we observe that these smaller areas have the form of rings (most of the time). Interestingly, in the very first moments of the evolution, a big square is formed on the surface, related

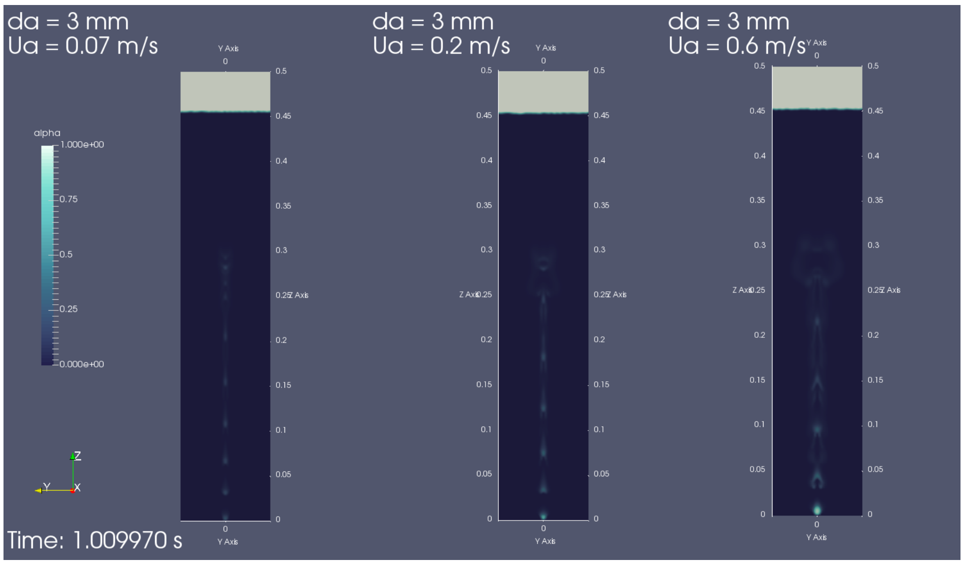


Figure 10: Phase distribution for the main case (center), that sets the inlet velocity of the air towards z to 0.3 m/s and for the same case with $U_{\mathrm{a},z} = 0.07$ m/s (left) and $U_{\mathrm{a},z} = 0.6$ m/s (right). When the air is injected with lower velocity, the bubble areas stay together, while in higher inlet velocities it tends to mix with the continuous phase. In all the cases, `da` is set to 3 mm.

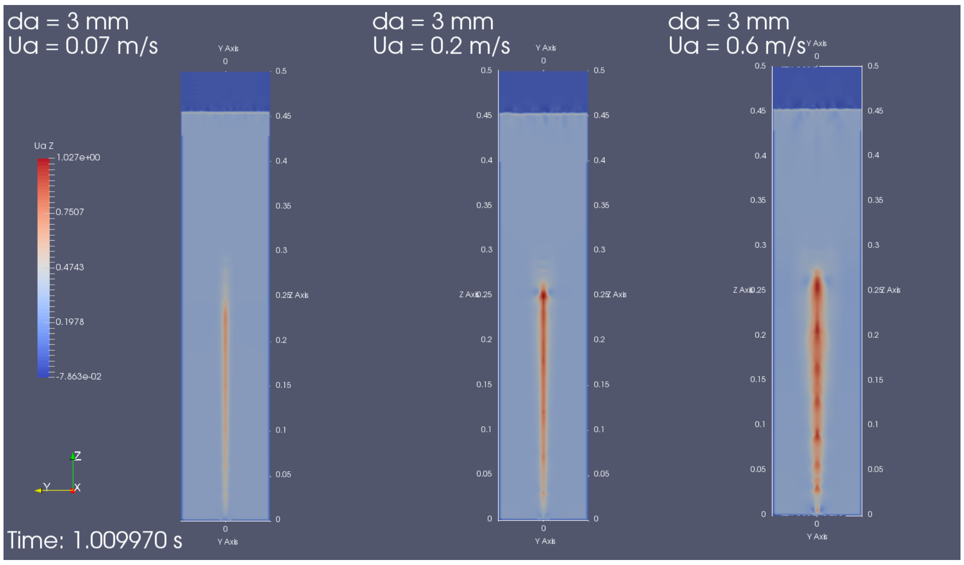


Figure 11: Dispersed phase velocity towards z for the main case (center), that sets the inlet velocity of the air towards z to 0.3 m/s and for the same case with $U_{\mathrm{a},z} = 0.07$ m/s (left) and $U_{\mathrm{a},z} = 0.6$ m/s (right). Higher inlet velocity leads also to higher air velocity inside the column. In all the cases, `da` is set to 3 mm.

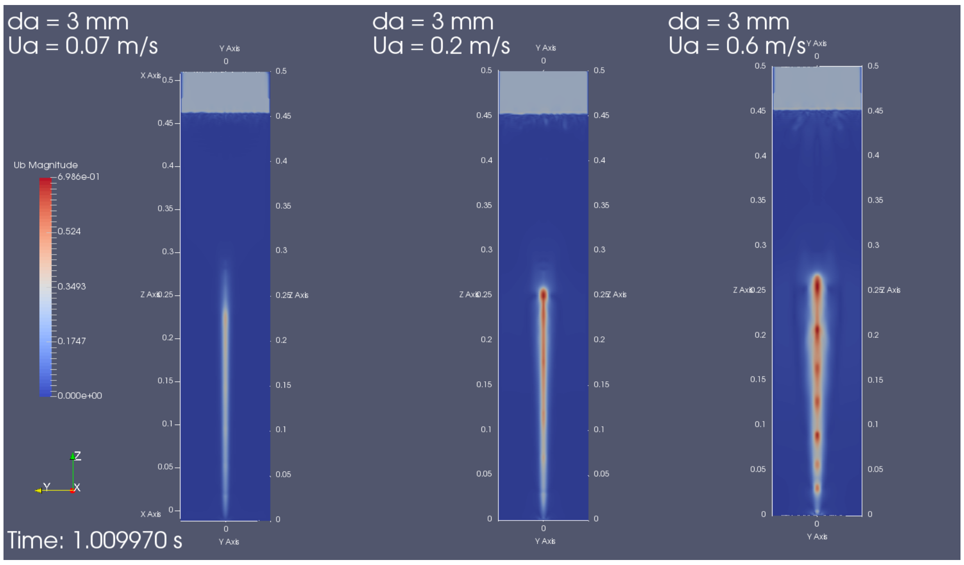


Figure 12: Continuous phase velocity towards z for the main case (center), that sets the inlet velocity of the air towards z to 0.3 m/s and for the same case with $U_{a,z} = 0.07$ m/s (left) and $U_{a,z} = 0.6$ m/s (right). Higher inlet velocity leads also to higher velocity inside for the continuous phase. In all the cases, `da` is set to 3 mm.

to the shape of the inlet hole (not shown here, see the animations).

4.5 Solver aspects

Some interesting problems and observations arose during our early experiments. In figure 15 an early low-velocity scenario is shown, with long air injections. Taking two slices on different height locations, we observe that the cross-section of the bubble has initially the shape of the square inlet. The shape changes to round as the bubble travels towards the surface. This hints us that the solver takes into account the surface tension and/or the pressure effects on the dispersed phase.

On figure 16 another problem that was observed in low-velocity scenarios with coarse grids is shown. In order to visualize bubbles in so small `alpha` values, very low thresholds are set, that allow us to observe the problem. Due to numerical issues, the outer cells of the inlet, in which the gradient is high, tend to give higher `alpha` values. This accumulates over time and looks like a (false) diffusion. The result is this unphysical, sharp face of the bubble.

On figure 17 `bubbleFoam` is compared to the newer `twoPhaseEulerFoam`, that is used in other studies. Overall the two solvers give very similar results. A case of difference is shown here, where the `twoPhaseEulerFoam` seems to

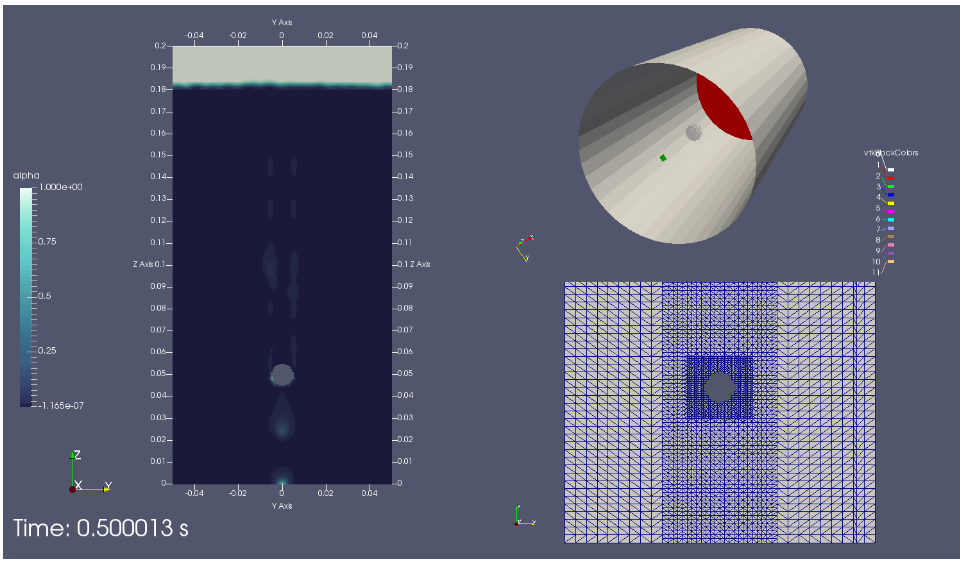


Figure 13: Interaction with an obstacle. **Left:** phase distribution. After colliding on the spherical object, the dispersed phase area is divided to two areas. **Right-top:** geometry of the scenario. The spherical obstacle is a part of the wall boundaries and it is aligned above the inlet. **Right-bottom:** Apart from the first stage of refinement, applied also in the main case (figure 3), a second level of refinement is applied around the obstacle. No surface-fitting was applied around the obstacle.

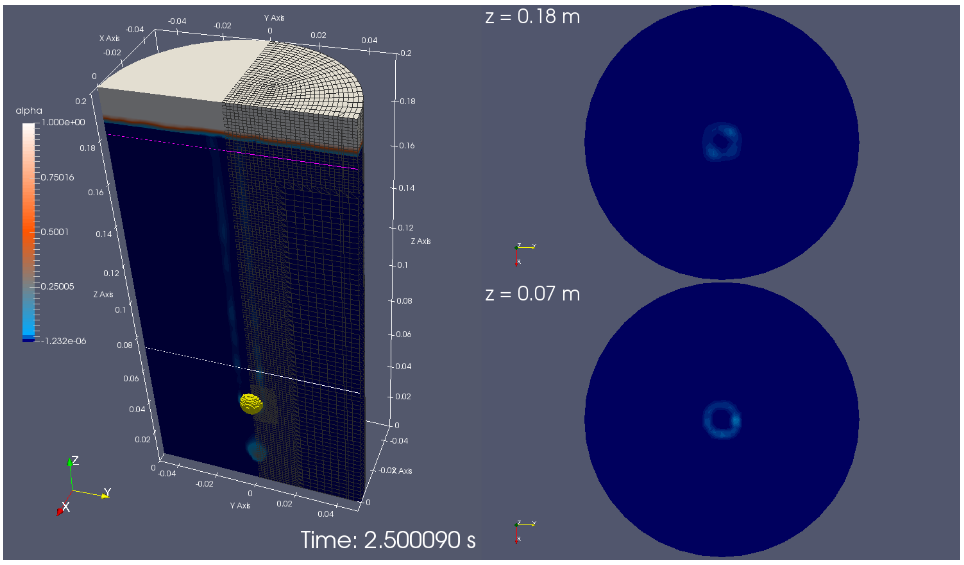


Figure 14: Interaction with an obstacle, 3D view and slices at $z = 0.07\,\mathrm{m}$ (bottom) and $z = 0.18\,\mathrm{m}$ (initial surface level, top). The colors are adjusted to better visualize the dispersed phase volume breakage. After the obstacle, the dispersed phase forms rings.

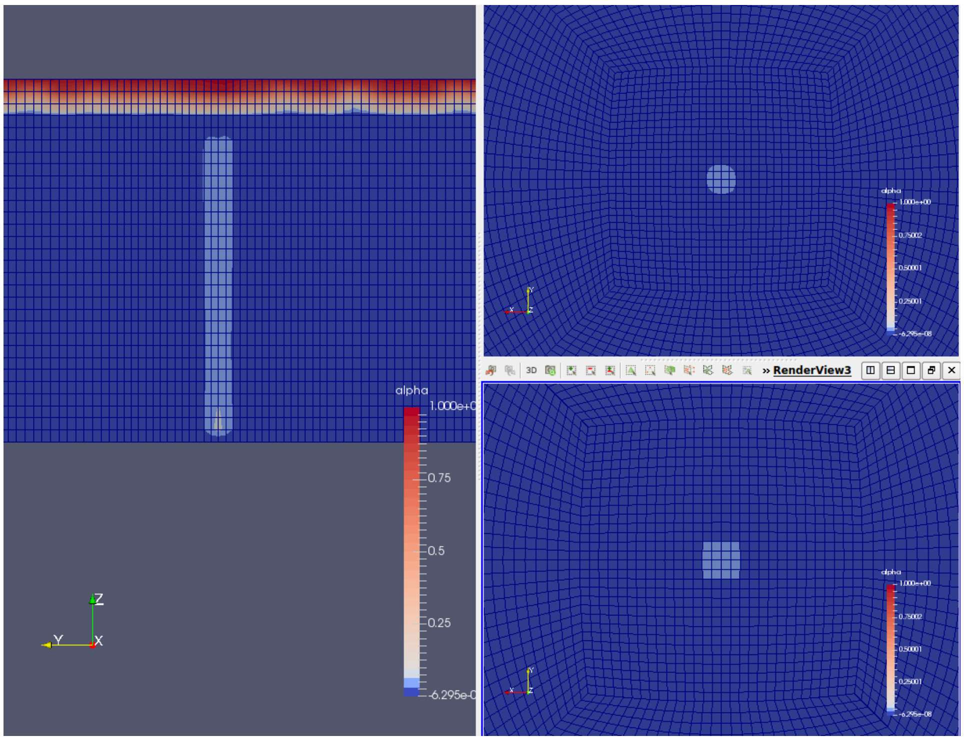


Figure 15: Early results with far from optimal parameters. In this low-velocity scenario (0.01 m/s, once every 0.1 s and for 0.05 s), the gradual transformation of the bubble cross-section from a perfect square (right-bottom) to a circle (right-top), as it travels towards the surface, can be observed. In our main case, this happens instantly and the gradual transformation cannot be observed. Note that the color scale has been adjusted to visualize the air-rich areas more clearly.

not interact with the surface in the same magnitude. The shown peak is getting higher over time, as more bubbles arrive, until it collapses. The shape of the bubbles is also a bit different on the bottom. Although the same parameters of `bubbleFoam` have been used in the `twoPhaseEulerFoam`, there are also some extra parameters that may affect the solution. They were set the same as in the provided tutorial.

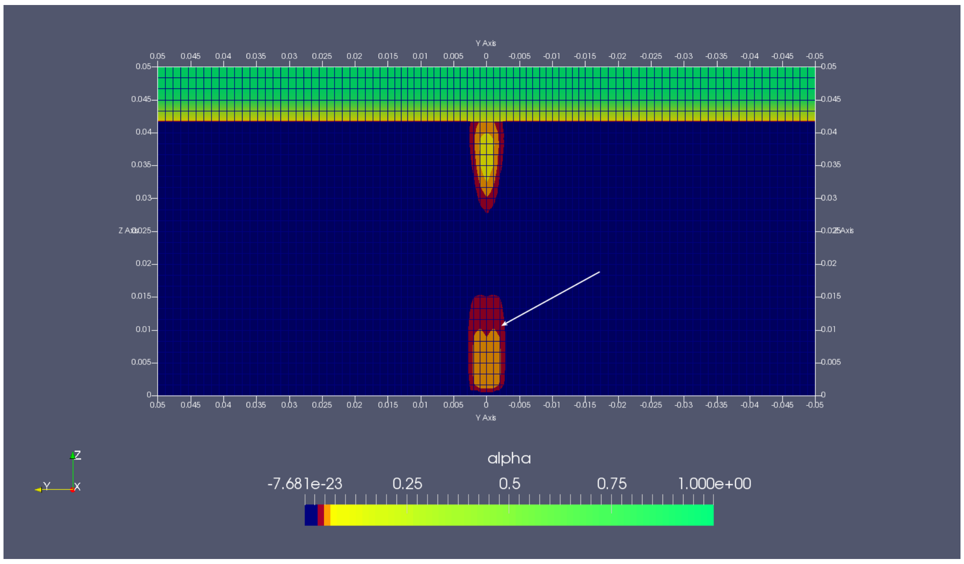


Figure 16: Similar early results as in figure 15. In order to visualize bubble areas, the color scale needed to be shifted towards very low `alpha` values. In these low values, numerical diffusion is observed, as the Volume-Of-Fluid method is used. Steep gradients are also introduced around the inlet, as the `Ua` is set uniformly for the respective inlet cells. Notice the sharp-edged shape of the injected bubble area. This was cured by increasing the inlet velocity and refining the mesh, therefore increasing the `alpha` values near the inlet.

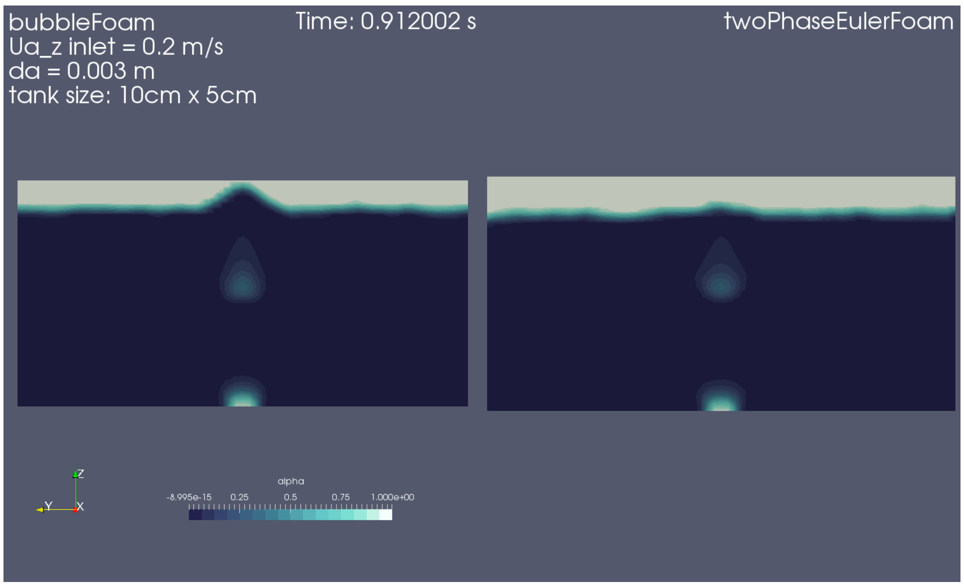


Figure 17: Comparison between the `bubbleFoam` and the newer `twoPhaseEulerFoam`. The *bubbleColumn* tutorial of each solver was used as a base, adapted to fit our approach. The same parameters of `bubbleFoam` were used in the `twoPhaseEulerFoam` case, although the latter takes also some extra parameters. The `bubbleFoam` affects more the surface, while the `twoPhaseEulerFoam` provides a rounder bubble volume. The observed peak is increasing in height over time and it collapses after a maximum.

5 Conclusion

5.1 Lessons learned

The figure 18 summarizes the tools we used at each step of our project. Overall, we combined several tools that we first used in the previous exercises, and we used some others for the first time, like the `bubbleFoam` and `twoPhaseEulerFoam` solvers, the `timeVaryingUniformFixedValue` boundary condition, the `sphereToCell` selection function and the `arc` type of edges in `blockMeshDict`. We didn't add any custom source code, as we tried to do everything with tools that were already available and therefore more generic and portable. We also used `Paraview` extensively, in order to better visualize our results.

We simulated a 3D bubble column scenario and we were able to observe the main phenomena that we expected, as well as some that we were not aware of before. Expected observations were the rising of the surface level, the water flow induced by the air flow, the interaction with the surface and the decrease in the size of the "bubbles" as they travel towards the top. We saw a clear dependency of the bubbles shape and velocity on their size and on the inlet conditions. We also observed the expected breakage of the bubble areas when they faced an obstacle. More interesting were the evolution of the bubble shape and especially the strong drag and lift effects that made the bubbles sharper, as well as their deceleration near the top. A nice addition to that was the strong cross-section transformation of the bubbles as they leave the square inlet hole.

Exploring the range of parameters in order to get results close to our physical expectations, we faced some problems and we better understood some numerical aspects. Since the beginning, we faced the challenge of simulating a big domain in feasible time and we learned that it is better to build the domain step-by-step. Therefore, we ran our first simulations with less cells, or on shorter columns with the same cell density. We were also assisted by the parallel execution feature of OpenFOAM, that in our problem was seamlessly configured. Some of the numerical issues that we faced was the selection of an appropriate timestep size for a stable solution, something that was solved using an adaptive timestep, bounded by the Courant number. We also saw that numerical diffusion is an important issue and we found some ways to heal it.

5.2 Future work

This project gave us an overview of how a bubble column simulation can be set and what aspects of the simulation are challenging. It helped us better understand the phenomena behind it, but it also opened many questions and ideas for further studies. The following would be interesting for us or maybe for other students too in the future:

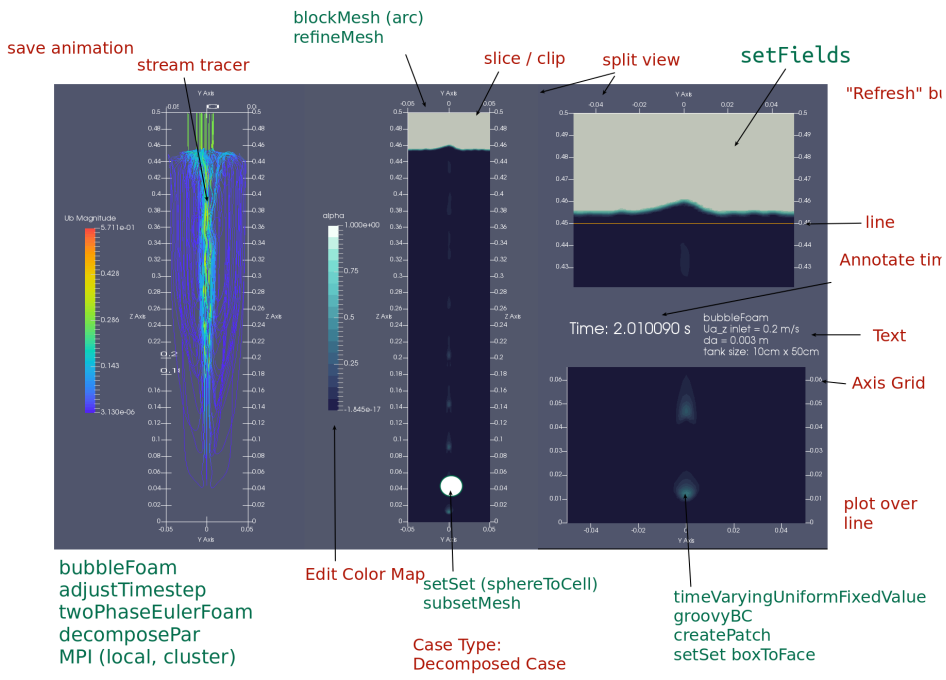


Figure 18: Overview of the tools used in this project.

1. Add more bubble injection points. This would require changes in the mesh and it would give us a better understanding of the complex phase interactions inside an actual bubble column. An inclusion of a range of phase diameters would also be a more realistic modelling approach.

2. Use a more advanced multiphase solver to study the phase interaction in better depth. Phenomena as heat and mass exchange could be added. An additional phase could be added, e.g. an oil phase, that could correspond to biomass. In that case, modelling of chemical reactions would be very interesting. There are several newer related solvers introduced in the newer versions of OpenFOAM, that include these features.

3. Add more obstacles and complex geometries. This would require the design of the geometry in a CAD software, conversion to OpenFOAM and adaption of the mesh. Objectives could be the maximization of the contact time between the two phases or the gradual breakage of big bubbles to finer. Systems like these are used, for example, in the aeration stage of wastewater treatment.

4. Add particle tracking. This could correspond to light solids that also exist in some of the related industrial problems.