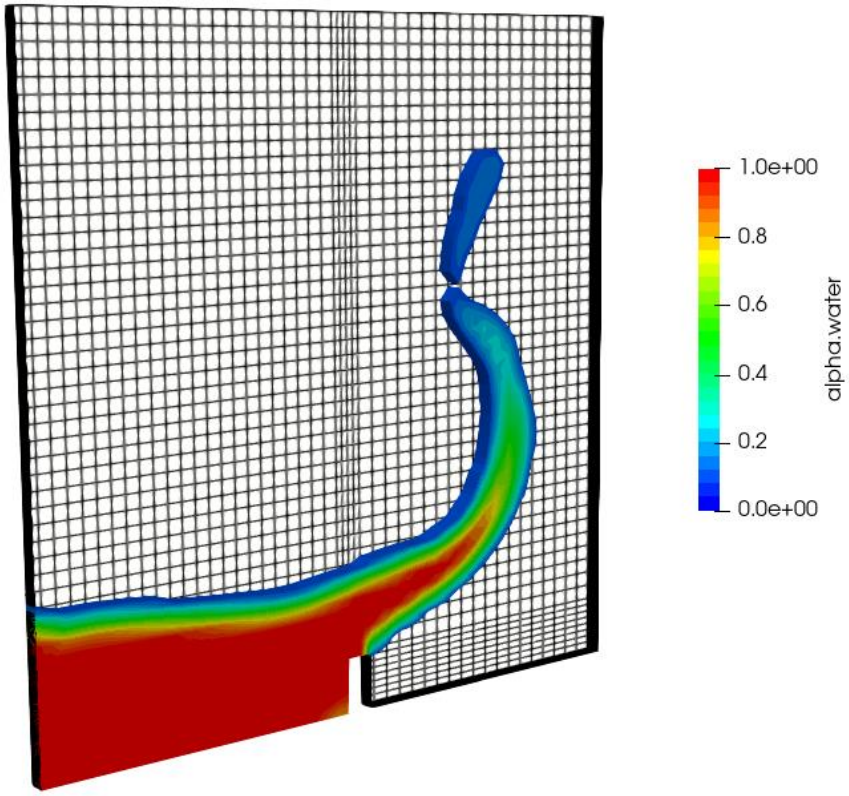
Tutorial Eight

Multiphase



Bahram Haddadi

ENGINEERING FLUID DYNAMICS

7th edition, March 2025

Contributors:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen
- Vikram Natarajan
- Jozsef Nagy

TU WIEN ICEBE IMAGINEERING NATURE

Technische Universität Wien

Institute of Chemical, Environmental

& Bioscience Engineering

Available from: www.fluiddynamics.at

Background

1. Multiphase flow

Multiphase flow refers to the simultaneous movement of two or more different phases (solid, liquid, or gas). Each phase may contain one or more chemical components. The common types of multiphase flows include:

- Gas-liquid flows (e.g., air bubbles in water, boiling liquids)
- Gas-solid flows (e.g., pneumatic transport, fluidized beds)
- Liquid-solid flows (e.g., sediment transport, slurry flows)
- Liquid-liquid flows (e.g., oil-water mixtures)
- Three-phase flows (e.g., water, gas, and solid mixtures in oil pipelines)

Multiphase flow can be further classified based on the flow regime:

- Separated flow: Distinct interfaces between phases (e.g., water flowing under oil).
- Dispersed flow: One phase exists as discrete particles or droplets within another continuous phase (e.g., bubbles in water, oil droplets in water).
- Mixed flow: A combination of separated and dispersed flow regions.

Multiphase flow is found in numerous applications, including chemical reactors (bubble columns, distillation towers), oil and gas industries (flow through pipelines, separation processes), environmental science (rain formation, erosion due to sediment transport) and power plants (boiling in nuclear reactors, cooling systems). Understanding and modeling multiphase flow is crucial for improving efficiency, design optimization, and operational safety in these applications.

2. Modeling approaches

Modeling of multiphase flow can be extremely complex, due to possible flow regime transitions. To simplify the matter, different modeling approaches can be adopted and they generally fall into two categories: lagrangian and Eulerian. In the case of dispersed configuration, Lagrangian approach is more suitable. This involves tracking individual point particles during its movement. The other approach is the Eulerian approach, which observes fluid behavior in a given control volume. Below we will cover some common modeling approaches of multiphase flow.

2.1. Euler-Euler approach (Multi-fluid model)

All phases are treated as continuous in the Euler-Euler approach. This approach is suitable for separated flows where each phase behaves as a continuum, rather than being discrete. The phases interact through the drag and lift forces acting between them, as well as through heat and mass transfer. The Euler-Euler approach is also capable of modeling dispersed flow, where

we are interested in the overall motion of particles rather than tracking individual particles.

In the Euler-Euler approach, we introduce the concept of phasic volume fractions. These fractions are assumed continuous functions of space and time, with their sum equal to one. For each phase, a set of conservation equations for mass, momentum and energy is solved individually; in addition, a transport equation for the volume fraction is solved. Coupling between the phases is achieved through shared pressure and interphase exchange coefficients.

2.2. Eddy Interaction Model

In the Eddy Interaction Model, each particle interacts with a succession of eddies. The fluid motion of the particle is characterized by three parameters: i) eddy velocity, ii) eddy lifetime, iii) eddy length. It follows the particle-tracking Lagrangian approach.

The eddy lifetime (t_e) and eddy length scale (l_e) are estimated from the local turbulence properties. From the length scale and the particle velocity, one can calculate the eddy transit time (t_c), i.e. the time taken for a particle to cross the eddy. The particle is then assumed to interact with the eddy for a time, which is the minimum of the eddy lifetime and the eddy transit time.

$$t_{int} = \min(t_e, t_c)$$

During that interaction, the fluid fluctuating velocity is kept constant and the discrete particle is moved with respect to its equation of motion. Then a new fluctuating fluid velocity is sampled and the process is repeated.

2.3. Volume of Fluid (VOF) method

VOF method belongs to the Eulerian class of modeling approach. It is based on the idea of **fraction function C**. Fraction function indicates whether a chosen phase is present inside the control volume. If C=1, the control volume is completely filled with the chosen phase; if C=0, the control volume is filled with a different phase. A value between 0 and 1 indicates that the interface between phases is present inside the control volume. It is important in VOF method that the flow domain is modeled on a fine grid, since the interface should be resolved.

The focus of the VOF method is to track the interface between phases. To do this, the transport equations are solved for mixture properties, assuming that all field variables are shared between the phases. Then an advection equation for the fraction function C is solved. The discretization of the fraction function equation is crucial for obtaining a sharp interface.

incompressibleVoF – damBreak

Tutorial outline

Use the incompressibleVoF solver to simulate breaking of a dam for 2s.

Objectives

- Understanding how to set viscosity, surface tension and density for two phases

Data processing

See the results in ParaView.

1. Pre-processing

1.1. Copying tutorial

Copy tutorial from the following folder to your working directory:

`$FOAM_TUTORIALS/incompressibleVoF/damBreak`

1.2. 0 directory

In the 0 directory, in the alpha.water.orig and p_rgh files, the initial values and boundary conditions for water phase and pressure are set. Copy alpha.water.orig to alpha.water (remember: the *.orig files are back up files, and solvers do not use them). E.g. in alpha.water:

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

dimensions      [];

internalField   uniform 0;

boundaryField
{
    #includeEtc  "caseDicts/setConstrainTypes"

    wall
    {
        type            zeroGradient;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * *//
```

`#includeEtc` includes the *etc* folder in the OpenFOAM installation directory and the path `"caseDicts/setConstrainTypes"` instruct it to include the boundaries in this file also here. In this case the empty boundary is used for this tutorial.
Checking the *blockMeshDict* file there is no boundary named wall, the `wall` in the `boundaryField` section of files in the *0* is for using the same boundary condition for the boundaries which have type wall.

Note: If in the files in 0 directory some not used boundaries are defined, as far as their syntax is correct, OpenFOAM will ignore them, so you don't need to remove them!

Note: In the `dimensions` section `[]` is equal to `[0 0 0 0 0 0 0]` and it means it is a dimensionless parameter.

Note: The `inletOutlet` and the `outletInlet` boundary conditions are used when the flow direction is not known. In fact, these are derived types and are a combination of two different boundary types.

- `inletOutlet`: When the flux direction is toward the outside of the domain, it works like a zeroGradient boundary condition and when the flux is toward inside the domain it is like a fixedValue boundary condition.
- `outletInlet`: This is the other way around, if the flux direction is toward outside the domain, it works like a fixedValue boundary condition and when the flux is toward inside the domain, it is like a zeroGradient boundary condition.

E.g. if the velocity field outlet is set as `inletOutlet` and the `inletValue` is set to `(0 0 0)`, it avoids backflow at the outlet! The “`inletValue`” or “`outletValue`” are values for `fixedValue` type of these boundary conditions and “`value`” is a dummy entery for OpenFOAM® for finding the variable type. Using `(0 0 0)`, OpenFOAM® understands that the variable is a vector.

1.3. constant directory

In the file phaseProperties, there is the list of phases in the simulation (in this case air and water):

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

Phases          (water air);

sigma           0.07;

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

and `sigma` is the surface tension between two phases, in this example it is the surface tension between air and water.

For each phase, there is a dedicated *physicalProperties.fileName* file, in which the properties of the relevant phase can be set. E.g. the *physicalProperties.water* file looks as following:

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

viscosityModel  constant;
nu              1e-06;
rho             1000;

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

In both phases the coefficients for different models of viscosity are given, e.g. `nu` and `rho`. Depending on which model is selected, the corresponding coefficients are read. In this simulation, the selected model is `constant` (representing Newtonian model), therefore only the `nu` coefficient is needed.

Checking the g file, the gravitational field and its direction are defined, it is `9.81` m/s^2 in the negative y direction.

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

dimensions      [0 1 -2 0 0 0 0];
value           ( 0 -9.81 0 );
```

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

1.4. system directory

In the *controlDict* change the `endTime` to `2`, for 2s of simulation.

2. Running simulation

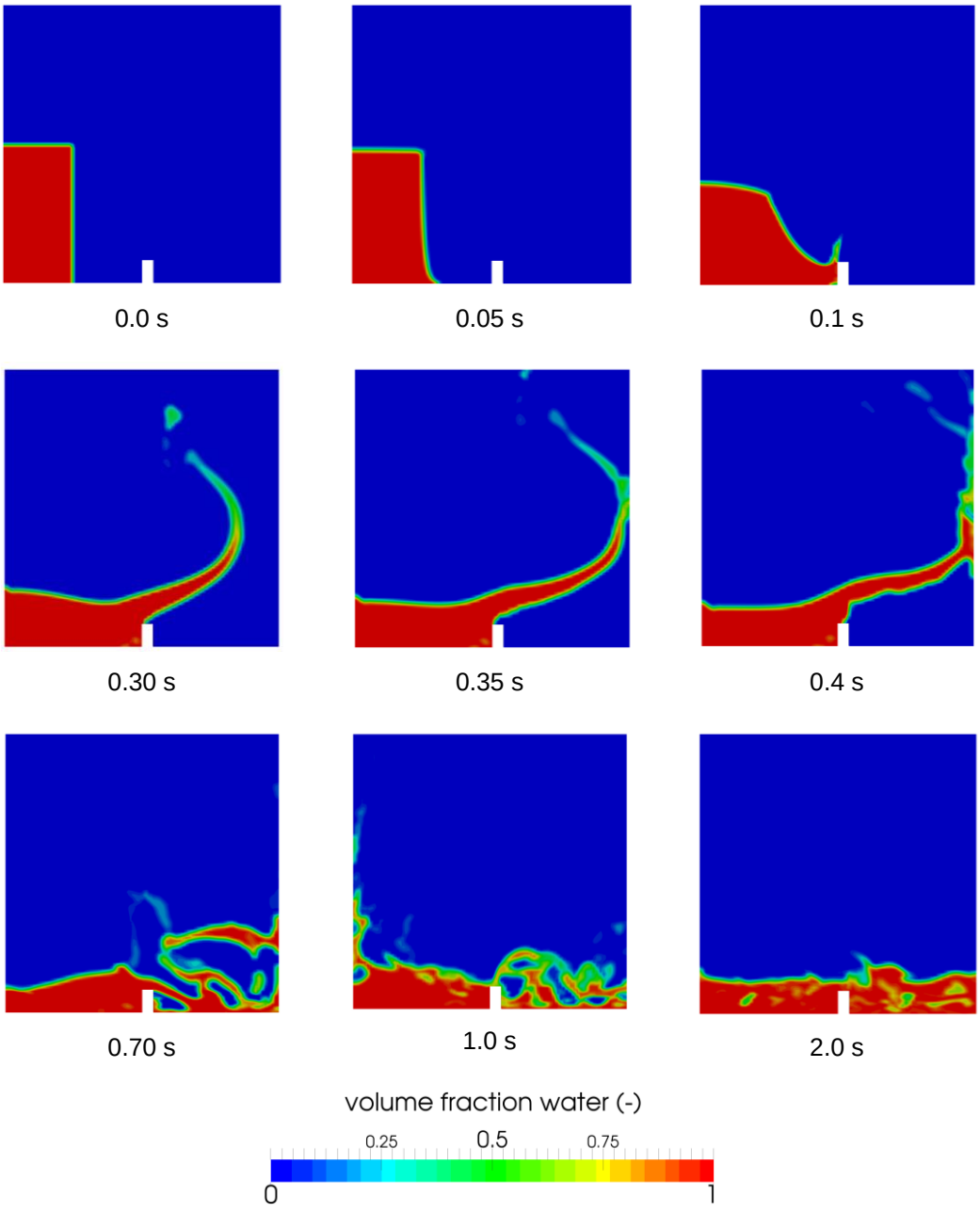
```
>blockMesh
```

```
>setFields
```

```
>foamRun -solver incompressibleVoF
```

3. Post-processing

The simulation results are as follows (these are not the results for the original mesh, but a 2x refined mesh):



Contours of the water volume fraction at different time steps