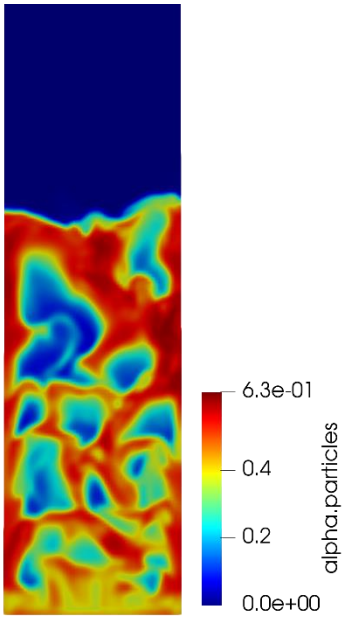
CEMF.ir
Center of Engineering Multiscale Modeling Fluid Flow

Mastering twoPhaseEulerFoam

One: Fluidized bed



How to simulate a gas-solid fluidized bed using OpenFOAM®

Compatible with

OpenFOAM® 7 | OpenFOAM® 6
OpenFOAM® 5.x | OpenFOAM® v1912

Authors

Hamidreza Norouzi and **Ramin Khodabandehlou**

Amirkabir University of Technology

Center of Engineering and Multiscale Modeling of Fluid Flow

License Agreement

This material is licensed under (CC BY-SA 4.0), unless otherwise stated.
https://creativecommons.org/licenses/by-sa/4.0/

This is a human-readable summary of (and not a substitute for) the license. Disclaimer.

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **Share alike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Notices:

- You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.
- No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Extra consideration:

- This document is developed to teach how to use OpenFOAM® software. The document has gone under several reviews to reduce any possible errors, though it may still have some. We will be glad to receive your comments on the content and error reports through this address: h.norouzi@aut.ac.ir

Document history

Revision	Description	Date
Rev1.2	Some corrections were made and tutorial was performed by OpenFoam 6 (Saied Mahdavi) and OpenFoam 5.x.	May, 9, 2020
Rev1.1	Simulation was performed using OpenFoam v1912.	April 30, 2020
Rev1.0	Reviewed to correct errors, setup files double checked.	April 28, 2020
Rev0	The first draft is prepared and run with OpenFOAM 7.	April 12, 2020

Table of Contents

Prerequisites 4
How to get simulation setup files? 4
1. Brief Description of twoPhaseEulerFoam 5
2. Problem definition 5
3. Simulation setup 6
3.1. Defining phases and interphase coupling parameters 6
3.1.1. Phases and phase definition 8
3.1.2. Interphase drag and Heat transfer 8
3.2. Turbulence properties of phases 10
3.2.1. Turbulence properties of particles (kinetic theory of granular flow) 10
3.2.2. Turbulence properties of air 12
3.3. Physical properties of phases 13
3.4. Gravitational acceleration 16
3.5. Generating geometry and mesh 16
3.6. Boundary and initial conditions 17
3.6.1. Velocity fields and granular temperature 17
3.6.2. Scalar fields 20
3.6.3. Creating initial packed bed of particles 20
4. Running the simulation 21
5. Results 21
Appendix A: A list of available thermoType settings for heRhoThermo model 24

Prerequisites

You need to be familiar with basics of OpenFoam® to start this tutorial.

How to get simulation setup files?

You have two options to get simulation setup files:

- **Tutorial cases:** execute the following command to copy one of the tutorial cases of OpenFoam® to the desktop of your computer. Then, you will need to make the necessary changes to the setup cases based on the instructions given in this tutorial.

```
> cp -r $FOAM_TUTORIALS/multiphase/twoPhaseEulerFoam/RAS/fluidisedBed $HOME/Desktop/fluidizedBed
```

- **Website:** setup files (a compressed file) are uploaded on www.cemf.ir alongside this tutorial file. You can find these files there.

1. Brief Description of twoPhaseEulerFoam

twoPhaseEulerFoam is a solver for a system of 2 non-reacting compressible fluid phases. One phase in this system is always dispersed. So it is a good candidate for simulating bubble columns in gas/liquid systems or fluidized beds and spouted beds in gas/solid systems. The solver also solves the energy equation (enthalpy or internal energy) for both phases and couple them by the one-film resistance heat transfer model.

In the case of gas-liquid systems, laminar and turbulence models (RAS and LES) models can be selected for both phases. In this case of gas-solid or liquid solid systems, these models can only be applied for the fluid phase. The solid phase momentum equation incorporates a model for the stress tensor. Two main approaches are possible: inviscid solid phase with a pressure model and a solid phase with stress tensor that is obtained by KTGF theory and frictional models.

Various sub-models for interphase coupling are also provided that make it possible to select a wide range of physical models for the system. It is also possible to extend the standard solver to include new sub-models in the simulation.

2. Problem definition

The fluidized bed is a flat fluidized bed with rectangular cross section with dimensions 150×28×2.5 cm^3. This bed is filled up to 40 cm with 275-μm spherical glass beads with the density of 2500 kg/m^3. The packing fraction (solid volume fraction) is assumed to be 0.6 and the maximum packing fraction was assumed to be 0.63. These particles are fluidized with air at ambient conditions (here we assume 1 atm and 283 K). Other operating conditions and simulation inputs are listed in Table 1. This condition exactly corresponds to the experiments performed by Taghipour et al. [Taghipour, Ellis, & Wong, 2005. Chemical Engineering Science, 60, 6857–6867].

Table 1: Simulation parameters and operating conditions

Particle mean diameter [μm]	275	Air viscosity [Pa.s]	1.84×10^{-8}
Particle density [kg/m3]	2500	Air density	Ideal gas
Particle heat Capacity [J/kg/K]	500,000	Air heat Capacity [J/kg/K]	1007
Maximum packing	0.63	Air Prandtl number	0.7
Restitution coefficient	0.95	Superficial gas velocity [m/s]	0.38
Packed bed height [m]	0.4	Inlet air temperature [K]	288
Packed bed fraction	0.6		
Particle-wall specularity coeff.	0.2		

3. Simulation setup

3.1. Defining phases and interphase coupling parameters

The simulation consists of two phases: air as the continuous phase and particles as the discrete phase. The settings for defining phases and their interphase coupling models can be found in constant/phaseProperties file. Various settings can be fixed in this file, though not all of them is required for this simulation. Only those settings that apply to this simulation are described here.

constant/phaseProperties

```
phases (particles air);

particles
{
    diameterModel constant;
    constantCoeffs
    {
        d           275.0e-6; // mean particle diameter
    }

    alphaMax        0.63;   // maximum allowable packing fraction
    residualAlpha   1e-6;
}

air
{
    diameterModel constant;
    constantCoeffs
    {
        d           1;
    }
    residualAlpha   0;
}

blending
{
    default
```

```
    {
        type            none;
        continuousPhase air;
    }
}

sigma
(
    (particles and air) 0
);

aspectRatio
(
);

drag
(
    (particles in air)
    {
        type            GidaspowErgunWenYu;
        residualRe      1e-3;
        swarmCorrection
        {
            type        none;
        }
    }
);

virtualMass
(
);

heatTransfer
(
    (particles in air)
    {
        type            RanzMarshall;
        residualAlpha   1e-3;
    }
);

lift
(
);

wallLubrication
(
);

turbulentDispersion
(
);

// Minimum allowable pressure
pMin            10000;
```

3.1.1. Phases and phase definition

`phases` entry defines two phase names for the simulation: `particles` and `air`. In `particles` sub-dictionary, the diameter model and residual alpha are specified. Here, `constant` diameter model with the mean diameter 275 μm and the maximum packing (`alphaMax`) of 0.63 are defined. Other diameter model options are available (see Table 2).

Table 2: List of diameter models implemented in twoPhaseEulerFoam

Diameter model	Description
`constant`	Assumes a constant diameter for the dispersed phase which can be defined in constantCoeffs dictionary. This can be used for both solid and fluid (gas/liquid) dispersed phases.
`isothermal`	Assumes an isothermal model for bubble diameter that changes directly with pressure. In isothermalCoeffs dictionary, two keywords d_0 and p_0 should be defined. d_0 represents mean bubble diameter at p_0 (in Pa). The mean bubble diameter at pressure p is given by: $d = d_0 (p_0/p)^{1/3}$
`IATE`	Interfacial Area Transport Equation for bubble diameter: It solves for the interfacial curvature per unit volume of the phase rather than interfacial area per unit volume. This model can only be used for bubbles as dispersed phase. More information can be found in [Ishii, M., Kim, S. and Kelly, J., Nuclear Engineering and Technology, 2005:37 (6)].

Air in the fluidized bed is always a continues phase. So, diameter model in the `air` sub-dictionary does not affect the results, though you are required to define it.

The dispersed and continuous phases are defined in `blending` sub-dictionary. Here, no blending model is applied and `air` is introduced as the `continuous` phase. In next tutorials in this series, you will see how to use other blending models in gas-liquid systems (bubble columns).

3.1.2. Interphase drag and Heat transfer

In this simulation the most important mechanism for interphase momentum transfer is drag force. In `drag` sub-dictionary, you need to define the phase pair for which the drag model is defined. In the phase pair – enclosed in double parenthesis, `(particles in air)` – the first one is the dispersed phase and the second is the continuous phase. Here, `GidaspowErgunWenYu` correlation is selected. Other drag models are listed in Table 3.

Table 3: list of the implemented drag models in twoPhaseEulerFoam solver

Drag model	More information
`Ergun`	Good for dense packing. More information in [ref1]
`Gibilaro`	For dilute to moderate packing. More information in [ref1]
`GidaspowErgunWenYu`	A combination of Ergun equation for dense packing $\alpha air \leq 0.8$ and WenYu correlation for loose packing $\alpha air > 0.8$. since the model switches from Ergun to WenYu at $\alpha air = 0.8$, it has discontinuous jump in drag force. More information in [Multiphase flow and fluidization, Gidaspow, D., Academic Press, New York, 1994].
`GidaspowSchillerNaumann`	It similar to SchillerNaumann model except it includes porosity correction function in drag coefficient function is $f = \alpha_{air}^{-2.65}$. So, it can be used for fluid-solid systems. However, applying it to very dense systems is not recommended. More information in [ref1].
`IshiiZuber`	Ishii and Zuber (1979) drag model is used for dense dispersed bubbly flows. More information in [Ishii, M., Zuber, N., AIChE Journal, 1979: 25 (5), 843-855].
`Lain`	This correlation is used for bubble columns. More information can be found in [Lain et al., International Journal of Multiphase Flow, 2002: 28 (8), 1381-1407.]
`SchillerNaumann`	Schiller and Naumann drag model is good for dispersed bubbly flows. Since it does not include the porosity correction function $f = 1$, it should not be applied for liquid-solid systems where particles are in the proximity of each other.
`SyamlalObrien`	[Syamlal et al. MFIX documentation, Theory Guide. Technical Note DOE/METC-94/1004. Morgantown, West Virginia, USA, 1993.]
`WenYu`	It is similar to GidaspowSchillerNaumann except the porosity correction in drag coefficient function is $f = \alpha_{air}^{-3.65}$. It is used for dilute to moderate packing.

ref1: Enwald et al., Int. J. Multiphase Flow, 1996.

In `heatTransfer` sub-dictionary, `RanzMarshall` model is selected for calculating heat transfer coefficient between `particles` as the dispersed phase and `air` as the continuous phase. With this setup, the heat transfer resistance is considered for the fluid phase (one-film resistance model).

The other option is `spherical` model which applies an analytical solution for heat transfer inside a sphere. This is particularly good when you are going to consider heat transfer resistance

in the solid particles (one-film resistance model) and ignore the heat transfer resistance in the fluid phase. In `spherical` model, Nu is constant and equal to 10.

Note

Alongside the implemented drag and heat transfer models in the standard release of OpenFOAM® for this solver, you can add new models for drag or heat transfer in this solver. This will be explained in part Two of this tutorial series.

3.2. Turbulence properties of phases

Turbulence properties of phases `air` and `particles` are separately defined in files turbuleceProperties.air and turbylenceProperties.particles.

3.2.1. Turbulence properties of particles (kinetic theory of granular flow)

In the file constant/turbulenceProperties.particles, you will find the model settings for particle phase rheology. Two main roots are available: `kineticTheory` and `phasePressure`. Since these two models are derived based on the RAS turbulence class, you need to choose `RAS` for the `simulationType`.

constant/turbulenceProperties.particles

```
simulationType  RAS;

RAS
{
    RASModel kineticTheory;

    turbulence      on;
    printCoeffs     on;

    kineticTheoryCoeffs
    {
        equilibrium             off;

        e                       0.95; // coefficient of restitution
        alphaMax                0.63; // maximum packing fraction
        alphaMinFriction        0.6;  // frictional stress is zero for lower values
        residualAlpha           1e-4;

        viscosityModel          Gidaspow;
        conductivityModel       Gidaspow;
```

```
        granularPressureModel   Lun;
        frictionalStressModel   JohnsonJackson;
        radialModel             SinclairJackson;

        JohnsonJacksonCoeffs
        {
            Fr                      0.05;
            eta                     2;
            p                       5;
            phi                     28.5;
            alphaDeltaMin           0.05;
        }
    }

    phasePressureCoeffs
    {
        preAlphaExp     500;
        expMax          1000;
        alphaMax        0.63;
        g0              1000;
    }
}
```

The simplest model is `phasePressure` in which no shear stress tensor is considered for the solid phase (solid phase is considered as an inviscid fluid). And the solid phase pressure is obtained by an exponential formula. The solid pressure becomes very high near maximum packing (α_{max}) preventing the solid phase from being packed beyond maximum packing. In this case, this model is not suitable. The parameters required for this model are shown in `phasePressureCoeffs` sub-dictionary.

The other model is `kineticTheory`, which is selected here. In the kinetic theory, the viscosity and hence the shear stress tensor of the solid phase is estimated by a property called granular temperature. Two main options are available for estimating granular temperature: algebraic relations and transport equation of granular temperature. You can select either of these by setting `equilibrium` to `on` or `off`. With setting `equilibrium` to `off`, the transport equation of granular temperature is solved and with setting `equilibrium` to `on`, algebraic relations are used for evaluating granular temperature.

There are some sub-models for evaluating solid conductivity, solid viscosity and pressure (viscous regime stress tensor) and frictional stress model (plastic regime pressure and viscosity

model) and radial function. Detailed information and applicability of each model can be found in the literature. The table below just lists the possible options to select among various models.

Table 4: Available models for the solid phase properties

Viscosity models	Conductivity models
`Syamlal`	`Syamlal`
`Gidospow`	`Gidospow`
`HrenyaSinclair`	`HrenyaSinclair`
Radial function models	Frictional stress models
`CarnahanStarling`	`JohnsonJackson`
`LunSavage`	`JohnsonJacksonSchaeffer`
`SinclairJackson`	`Schaeffer`
Granular pressure models	
`Lun`	`SyamlalRogersOBrien`

In `kineticTheoryCoeffs` sub-dictionary, `alphaMax` represent the maximum packing fraction and `alphaMinFriction` represents the minimum solid volume fraction for which frictional stresses are zero. Frictional model `schaeffer` uses `alphaMinFriction` as the critical volume fraction in its calculations and frictional model `JohnsonJackson` uses both `alphaMax` and `alphaMinFriction` in its calculations.

Note that some of these models require extra parameters. For example, `JohnsonJackson` friction model requires `Fr`, `eta`, `p`, `phi`, and `alphaDeltaMin` to be defined in `JohnsonJacksonCoeffs` dictionary. The first three parameters are model constants and `phi` is angle of internal friction in degrees and `alphaDeltaMin` is the minimum value for α-α_{max}. Similarly, `Schaeffer` friction model requires phi to be defined in `SchaefferCoeffs` dictionary (if it was selected).

3.2.2. Turbulence properties of air

In file constant/turbulenceProperties.air, you can define the turbulence model for the gas phase. Here, the k-ε turbulence model with default model coefficient is selected for air.

constant/turbulenceProperties.air

```
simulationType  RAS;

RAS
{
    RASModel kEpsilon;

    turbulence      on;
    printCoeffs     on;
}
```

In addition to RSA-type turbulence models, other turbulence models are also implemented for `twoPhaseEulerFoam` solver. The table below (Table lists all the available models for laminar, RAS and LES turbulence model types.

Table 5: Available turbulence models in twoPhaseEulerFoam solver

Laminar	`Stokes`
RAS	`kEpsilon`
	`kOmegaSST`
	`kOmegaSSTSato`
	`mixtureKEpsilon`
	`LaheyKEpsilon`
	`continuousGasKEpsilon`
LES	`Smagorinsky`
	`kEqn`
	`SmagorinskyZhang`
	`NicenoKEqn`
	`continuousGasKEqn`

Note

It is also possible to add new turbulence models other than those already implemented for this solver. This will be covered in part Two of this tutorial series.

3.3. Physical properties of phases

Physical properties of phases are defined separately in thermophysicalProperties.air and thermophysicalProperties.particles in constant folder. OpenFOAM® provides a bunch of models for modeling density, enthalpy and transport properties of fluids. The details of these thermophysical models is not covered in this tutorial. The properties of air are defined as follows:

constant/thermophysicalProperties.air

```
thermoType
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState perfectGas;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        molWeight   28.9;
    }
    thermodynamics
    {
        Cp          1007;     // heat capacity
        Hf          0;        //
    }
    transport
    {
        mu          1.84e-05;  //Pa.s
        Pr          0.7;
    }
}
```

In `thermoType` sub-dictionary, you can select the type of physical model the phase. For this phase, air, ideal gas EOS is selected for density, constant heat capacity model (`hConst`) for enthalpy, and constant transport properties for dynamic viscosity and heat conductivity (it is calculated using Prandtl number). The thermophysical `type` is `heRhoThermo` that means properties are calculated based on the enthalpy/internal energy and density of the mixture. And finally, entry `energy` defines the type of energy equation you want to be solved for this phase. Two options are available: `sensibleInternalEnergy` and `sensibleEnthalpy`. The first option tells the solver to solve energy equation based on the sensible internal energy of the phase and the second, based on the sensible enthalpy.

The properties of particles phase are defined as follows:

constant/thermophysicalProperties.particles

```
thermoType
{
    type            heRhoThermo;
    mixture         pureMixture;
    transport       const;
    thermo          hConst;
    equationOfState rhoConst;
    specie          specie;
    energy          sensibleInternalEnergy;
}

mixture
{
    specie
    {
        molWeight   100;
    }
    equationOfState
    {
        rho         2500; // kg/m3
    }
    thermodynamics
    {
        Cp          500000; // J/kg/K
        Hf          0;
    }
    transport
    {
        mu          0;
        Pr          1;
    }
}
```

Here, a constant density is assumed for solid particles which is 2500 kg/m^3. The dynamic viscosity, `mu`, is set to zero. Note that the stress tensor of the solid phase is calculated from KTGF and this value does not affect the momentum equation. It can also affect the effective conductivity (`kappa`/`Cp`) for enthalpy equation – note that conductivity for enthalpy equation is different from conductivity for temperature transport equation, which is denoted by `kappa`. Effective conductivity of enthalpy is the sum of two terms: laminar and turbulent conductivities. The laminar part is calculated from the values you supply in this file: calculated as `mu/Pr`. So with these settings, the laminar part is zero and the turbulent part (obtained form KTGF) would be considered only.

3.4. Gravitational acceleration

The gravitational acceleration can be defined in constant/g file as follows:

constant/g

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

3.5. Generating geometry and mesh

A 3D simulation of the fluidized bed will be performed here. `blockMesh` utility will be used to generate the geometry and hexagonal mesh. The edge dimensions of cells should be larger than the mean particle diameter to ensure that the averaged Navier-Stokes equations are valid in this simulation. In this simulation the cell edge size is 0.5 cm, which is almost 18 times the particle diameter. Settings to run `blockMesh` utility can be found in blockMeshDict file located in the constant folder.

constant/blockMeshDict

```
vertices
(
    (0.00 0.0 0.00 )
    (0.28 0.0 0.00 )
    (0.28 1.5 0.00 )
    (0.00 1.5 0.00 )
    (0.00 0.0 0.025)
    (0.28 0.0 0.025)
    (0.28 1.5 0.025)
    (0.00 1.5 0.025)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (56 300 5) simpleGrading (1 1 1)
);
patches
(
    patch inlet
    (
        (1 5 4 0)
    )
    patch outlet
    (
        (3 7 6 2)
    )
    wall walls
    (
        (0 4 7 3)
        (2 6 5 1)
```

```
        (0 3 2 1)
        (4 5 6 7)

    )
);
```

Execute the `blockMesh` command from the case directory. Three boundary patches are created.

```
> blockMesh
```

3.6. Boundary and initial conditions

Initial and boundary conditions of the filed variables are defined in 0 folder. Here, we give a brief overview of the important boundary conditions.

3.6.1. Velocity fields and granular temperature

In the file U.air, boundary conditions of the air velocity filed are specified. For the all side walls, `noSlip` condition is applied. For the inlet, `interstitialInletVelocity` is applied. In a gas fluidized bed, superficial gas velocity is known, which is the average velocity of gas when the bed is empty (the volumetric gas flow divided by bed cross section area). In this simulation, both air and particles exist in the inlet port and the void fraction of these phases changes during time. With a `fixedValue` boundary condition for the air, a constant superficial gas flow in the inlet won't be achieved. Therefore, `interstitialInletVelocity` is applied to give the superficial gas velocity for air. In the outlet, `pressureInletOutletVelocity` is used to make sure that the boundary condition switches between `zeroGradient` and `fiexdValue` when a reverse flow occurs.

0/ U.air

```
internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type               interstitialInletVelocity;
        inletVelocity      uniform (0 0.38 0);
        alpha              alpha.air;
        value              $internalField;
    }

    outlet
```

```
    {
        type                pressureInletOutletVelocity;
        phi                 phi.air;
        value               $internalField;
    }

    walls
    {
        type            noSlip;
    }

}
```

In the file U.particles, boundary conditions for velocity filed of particles are specified. Particles do not cross the inlet and outlet ports, so the velocity of the particle phase is set to zero. For side walls, all the boundary conditions can be applied: `slip`, `noSlip`, `zeroGradient`, and etc. However, in the case of granular flow, particles do not stick to the walls (no-slip conditions) and do not slide freely on the wall (slip conditions). They behave between these two conditions. Jahnson & Jackson [Johnson & Jackson, 1987. Journal of Fluid Mechanics, 176, 67–93] proposed a boundary conditions for solid phase velocity and granular temperature which read as:

$$\mu_s \frac{\partial u_s}{\partial x} = -\frac{\pi\, \phi_s \rho_s \alpha_s g_0 \sqrt{\theta_s}}{2\sqrt{3}\alpha_s^{max}} u_s$$

$$\kappa_s \frac{\partial \theta_s}{\partial x} = -\frac{\pi\, \phi_s u_s^2 \rho_s \alpha_s g_0 \sqrt{\theta_s}}{2\sqrt{3}\alpha_s^{max}} - \frac{\pi\sqrt{3}\rho_s \alpha_s g_0 (1 - e_W^2)\sqrt{\theta_s}}{4\alpha_s^{max}} \theta_s$$

where μ_s and κ_s are viscosity and conductivity of solid phase, ϕ_s and e_W are specularity coefficient and particle-wall coefficient of restitution.

This boundary condition is selected on the walls for U.particles and Theta.particles fields. You need to define the specularity coefficient and coefficient of restitution for particle-wall contacts when using Johnson & Jackson boundary conditions. Both coefficients should be between zero and one.

0/ U.particles

```
internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type               fixedValue;
        value              uniform (0 0 0);
    }

    outlet
    {
        type               fixedValue;
        value              uniform (0 0 0);
    }

    walls
    {
        type                    JohnsonJacksonParticleSlip;
        value                   $internalField;
        specularityCoefficient  0.2;
    }
}
```

You see the content of 0/Theta.particles file for granular temperature. Note that in this file, `referenceLevel` is specified. When this value is specified, it is initially added to the value of the field in all cells and boundary patches for time zero. In this way, the value of granular temperature will be non-zero in all cells and boundary conditions and this will prevent invalid math operations on granular temperature like division by zero.

0/Theta.particles

```
internalField       uniform 0;

referenceLevel      1e-4;

boundaryField
{
    inlet
    {
        type               fixedValue;
        value              uniform 1e-4;
    }

    outlet
    {
        type               zeroGradient;
    }

    walls
    {
        type                    JohnsonJacksonParticleTheta;
        value                   $internalField;
```

```
        restitutionCoefficient  0.95;
        specularityCoefficient  0.2;
    }

}
```

3.6.2. Scalar fields

For temperature fields of both air and particles phases, `zeroGradient` conditions are applied for the walls and `inletOutlet` condition for the outlet. The initial and inlet temperature for both phases are 288 K.

Since the mesh is not resolved enough near the walls, standard wall functions are applied for air turbulence fields, k.air, epsilon.air, and nut.air: `kqRWallFunction`, `epsilonWallFunction`, and `nutkWallFunction`. For k.air and epsilon.air, `inletOutlet` condition is applied for the outlet and `fixedValue` for the inlet.

For phase fraction field of both phases, alpha.air and alpha.particles, `zeroGradient` conditions are applied for all boundary patches.

Note

Since the original setup files were for a 2D simulation, a `frontAndBack` boundary was defined in the blcokMeshDict and for all fields in folder 0. Here, you are simulating a 3D fluidized bed with no `frontAndBack` boundary (as empty boundary). Make sure that you don't see any `frontAndBack` boundary defined in all files in folder 0.

3.6.3. Creating initial packed bed of particles

Based on the problem definition, the bed is initially filled by solid particles up to 40 cm height with packing density of 0.6 and hence the upper part of the bed is empty (α_{air} = 1). These conditions (non-uniform initial conditions) can be specified by using `setFields` utility. The settings can be found in system/setFieldsDict. The `default` settings are applied for the whole domain (bed) first and then the values specified in the regions dictionary overwrite the default values. Here, the default settings are α_{air} = 1 and $\alpha_{particles}$ = 0, which create an empty bed. Then,

volume fractions of the cells that reside in the `box` defined by two points `(0 0 0.0)` and `(0.28 0.4 0.025)` are set to α_{air} = 0.4 and $\alpha_{particles}$ = 0.6. Execute the `setFields` command from case directory to apply these initial conditions.

system/setFieldsDict

```
defaultFieldValues
(
    volScalarFieldValue alpha.air 1
    volScalarFieldValue alpha.particles 0
);

regions
(
    boxToCell
    {
        box (0 0 0.0) (0.28 0.4 0.025);
        fieldValues
        (
            volScalarFieldValue alpha.air 0.4
            volScalarFieldValue alpha.particles 0.6
        );
    }
);
```

Note

If you are using OpenFoam v1912 and before executing `setFields`, you must first copy and rename alpha.air.orig and alpha.particles.orig to alpha.air and alpha.particles.

4. Running the simulation

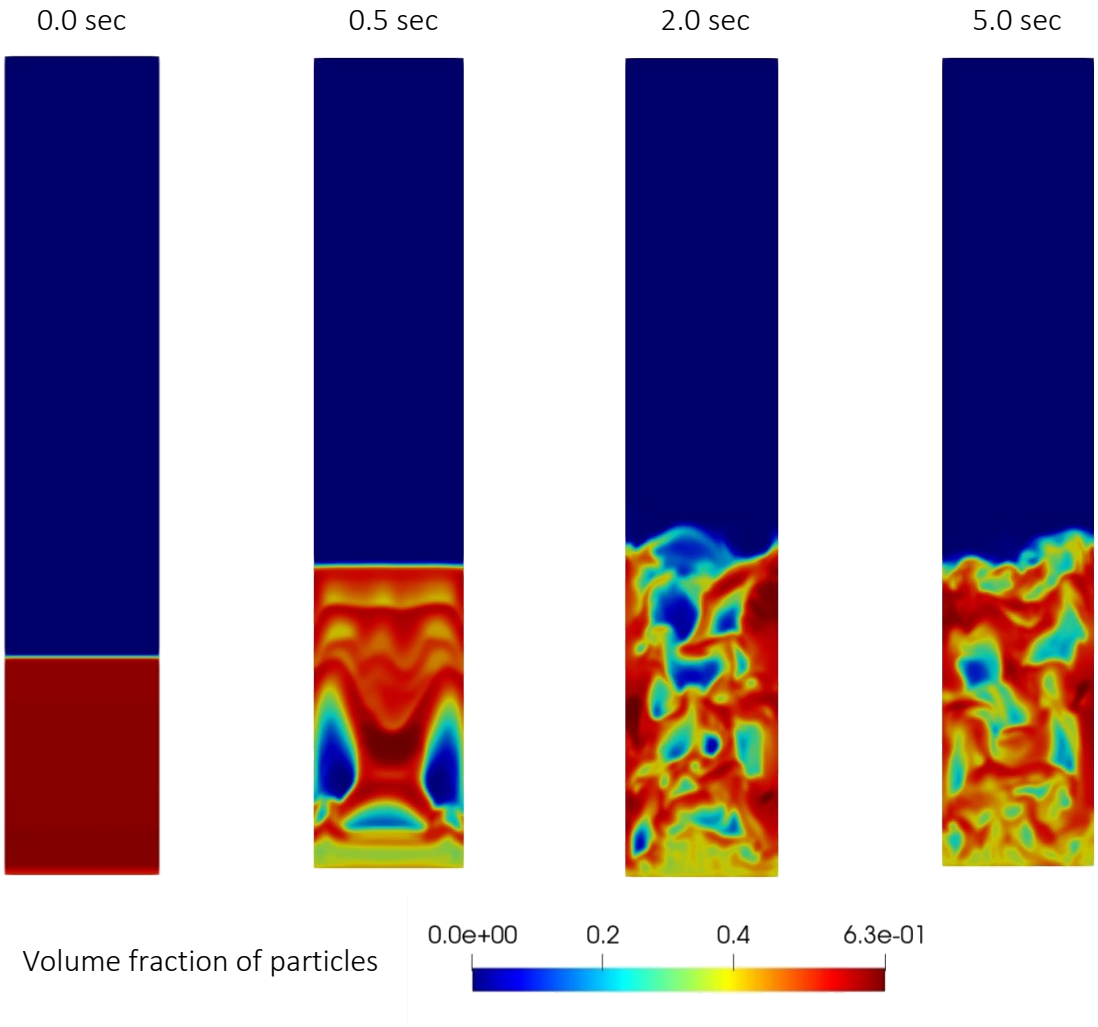
You need to execute the following commands in the case directory:

```
> blockMesh
> setFields
> twoPhaseEulerFoam
```

Depending on the computational power of your computer, it may take several minutes to complete the simulation.

5. Results

Some snapshots of the fluidized bed simulation are presented here.


0.0 sec
0.5 sec
2.0 sec
5.0 sec
Volume fraction of particles
0.0e+00
0.2
0.4
6.3e-01

Appendix A: A list of available thermoType settings for heRhoThermo model

mixture	transport	thermo	equationOfState	energy
homogeneousMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
homogeneousMixture	const	hConst	perfectGas	sensibleEnthalpy
homogeneousMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
homogeneousMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
inhomogeneousMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
inhomogeneousMixture	const	hConst	perfectGas	sensibleEnthalpy
inhomogeneousMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
inhomogeneousMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
multiComponentMixture	const	eConst	adiabaticPerfectFluid	sensibleInternalEnergy
multiComponentMixture	const	eConst	incompressiblePerfectGas	sensibleInternalEnergy
multiComponentMixture	const	eConst	perfectFluid	sensibleInternalEnergy
multiComponentMixture	const	eConst	perfectGas	sensibleInternalEnergy
multiComponentMixture	const	eConst	rhoConst	sensibleInternalEnergy
multiComponentMixture	const	hConst	adiabaticPerfectFluid	sensibleEnthalpy
multiComponentMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
multiComponentMixture	const	hConst	perfectFluid	sensibleEnthalpy
multiComponentMixture	const	hConst	perfectGas	sensibleEnthalpy
multiComponentMixture	const	hConst	rhoConst	sensibleEnthalpy
multiComponentMixture	polynomial	hPolynomial	icoPolynomial	sensibleEnthalpy
multiComponentMixture	polynomial	hPolynomial	icoPolynomial	sensibleInternalEnergy
multiComponentMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
multiComponentMixture	sutherland	janaf	incompressiblePerfectGas	sensibleInternalEnergy
multiComponentMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
multiComponentMixture	sutherland	janaf	perfectGas	sensibleInternalEnergy
pureMixture	WLF	eConst	rhoConst	sensibleInternalEnergy
pureMixture	const	eConst	Boussinesq	sensibleInternalEnergy
pureMixture	const	eConst	adiabaticPerfectFluid	sensibleInternalEnergy
pureMixture	const	eConst	perfectFluid	sensibleInternalEnergy
pureMixture	const	eConst	rhoConst	sensibleInternalEnergy
pureMixture	const	hConst	Boussinesq	sensibleEnthalpy
pureMixture	const	hConst	Boussinesq	sensibleInternalEnergy
pureMixture	const	hConst	adiabaticPerfectFluid	sensibleEnthalpy
pureMixture	const	hConst	adiabaticPerfectFluid	sensibleInternalEnergy
pureMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
pureMixture	const	hConst	incompressiblePerfectGas	sensibleInternalEnergy
pureMixture	const	hConst	perfectFluid	sensibleEnthalpy
pureMixture	const	hConst	perfectFluid	sensibleInternalEnergy
pureMixture	const	hConst	perfectGas	sensibleEnthalpy
pureMixture	const	hConst	perfectGas	sensibleInternalEnergy
pureMixture	const	hConst	rhoConst	sensibleEnthalpy
pureMixture	const	hConst	rhoConst	sensibleInternalEnergy

pureMixture	polynomial	hPolynomial	PengRobinsonGas	sensibleEnthalpy
pureMixture	polynomial	hPolynomial	icoPolynomial	sensibleEnthalpy
pureMixture	polynomial	hPolynomial	icoPolynomial	sensibleInternalEnergy
pureMixture	polynomial	janaf	PengRobinsonGas	sensibleEnthalpy
pureMixture	sutherland	hConst	Boussinesq	sensibleEnthalpy
pureMixture	sutherland	hConst	Boussinesq	sensibleInternalEnergy
pureMixture	sutherland	hConst	PengRobinsonGas	sensibleEnthalpy
pureMixture	sutherland	hConst	incompressiblePerfectGas	sensibleEnthalpy
pureMixture	sutherland	hConst	incompressiblePerfectGas	sensibleInternalEnergy
pureMixture	sutherland	hConst	perfectGas	sensibleEnthalpy
pureMixture	sutherland	hConst	perfectGas	sensibleInternalEnergy
pureMixture	sutherland	janaf	Boussinesq	sensibleEnthalpy
pureMixture	sutherland	janaf	Boussinesq	sensibleInternalEnergy
pureMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
pureMixture	sutherland	janaf	incompressiblePerfectGas	sensibleInternalEnergy
pureMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
pureMixture	sutherland	janaf	perfectGas	sensibleInternalEnergy
reactingMixture	const	eConst	adiabaticPerfectFluid	sensibleInternalEnergy
reactingMixture	const	eConst	incompressiblePerfectGas	sensibleInternalEnergy
reactingMixture	const	eConst	perfectFluid	sensibleInternalEnergy
reactingMixture	const	eConst	perfectGas	sensibleInternalEnergy
reactingMixture	const	eConst	rhoConst	sensibleInternalEnergy
reactingMixture	const	hConst	adiabaticPerfectFluid	sensibleEnthalpy
reactingMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
reactingMixture	const	hConst	perfectFluid	sensibleEnthalpy
reactingMixture	const	hConst	perfectGas	sensibleEnthalpy
reactingMixture	const	hConst	rhoConst	sensibleEnthalpy
reactingMixture	polynomial	hPolynomial	icoPolynomial	sensibleEnthalpy
reactingMixture	polynomial	hPolynomial	icoPolynomial	sensibleInternalEnergy
reactingMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
reactingMixture	sutherland	janaf	incompressiblePerfectGas	sensibleInternalEnergy
reactingMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
reactingMixture	sutherland	janaf	perfectGas	sensibleInternalEnergy
singleStepReactingMixture	sutherland	janaf	perfectGas	sensibleEnthalpy
singleStepReactingMixture	sutherland	janaf	perfectGas	sensibleInternalEnergy
veryInhomogeneousMixture	const	hConst	incompressiblePerfectGas	sensibleEnthalpy
veryInhomogeneousMixture	const	hConst	perfectGas	sensibleEnthalpy
veryInhomogeneousMixture	sutherland	janaf	incompressiblePerfectGas	sensibleEnthalpy
veryInhomogeneousMixture	sutherland	janaf	perfectGas	sensibleEnthalpy