blastFoam Theory and User Guide

Version 4.0.0

Jeff Heylmun, Peter Vonk
and Timothy Brewer
Synthetik Applied Technologies
info@synthetik-technologies.com

https://github.com/synthetik-technologies/blastFoam

April 13, 2020

Abstract

blastFoam is an open-source toolbox for simulating detonations based on the OpenFOAM®[1] framework (OpenCFD Ltd., 2018a). *blastFoam* provides solutions to highly compressible systems including single- and multi-phase compressible flow based on the work of Zheng et al. (2011), Shyue (2001), and Houim and Oran (2016). *blastFoam* provides implementations of the essential numerical methods (e.g. 2nd and 3rd order schemes), equations of state (e.g. ideal gas, stiffened gas, Jones-Wilkins-Lee, etc.), run-time selectable flux schemes (e.g. HLL, HLLC, AUSM+, Kurganov/Tadmor) and high-order explicit time integration (e.g. 2nd, 3rd, and 4th order). *blastFoam* provides activation and explosive burn models to simulate the initiation and expansion of energetic materials, as well as afterburn models to simulate under-oxygenated explosives that exhibit delayed energy release.

[1] *DISCLAIMER: This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software via www.openfoam.com, and owner of the OPENFOAM and OpenCFD trade marks.*

Contents

1 Introduction

blastFoam is an opensource computational fluid dynamics (CFD) code based on the OpenFOAM C++ library (OpenCFD Ltd., 2018a) to simulate strongly compressible systems of equations with detonations. Along with the main solvers for simulating fluid flow, additional utilities have been added to simplify case setup, allow adaptive mesh refinement with load balancing based on the work of Rettenmaier et al. (2019), and facilitate *blastFoam*-specific case post-processing.

The purpose of this guide is to serve as a reference for the governing equations and the models that have been implemented in *blastFoam* . The guide also provides a brief description of the capabilities of the solver and the new utilities that have been introduced on top of those available in the standard OpenFOAM.

1.1 System Requirements

blastFoam currently builds against OpenFOAM-7[2]. A modified version of OpenFOAM-7 with several bug fixes that are not included in the main release can be found at https://github.com/synthetik-technologies/OpenFOAM-7. The following dependencies are required to successfully compile and run *blastFoam*:

1. A working installation of OpenFOAM-7 complete with libraries and headers produced by OpenFOAM-7, no other packages are required to run *blastFoam* applications. **NOTE:** The precompiled versions of OpenFOAM-7 available for download include a bug which cases the use of AMR to fail. This bug has been fixed on the most recent version on github so it is recommend to compile from the source files. Additionally a forked version of OpenFOAM-7 is available at https://github.com/synthetik-technologies/OpenFOAM-7 that includes additional bug fixes necessary for more advanced features in *blastFoam* (such as moving meshes).

2. In order to create and view the postscript plots (e.g. *.eps files) created in the validation cases, *gnuplot* and an eps viewer must be installed.

[2] https://github.com/OpenFOAM/OpenFOAM-7

gnuplot is available on all platforms. EPS Viewer can be used on Windows and *ghostview* can be used on linux. On macOS, *Preview* opens eps files natively.

1.2 Downloading

The un-compiled source code can be obtained at: https://github.com/synthetik-technologies/blastfoam. Download or clone the source in the following location: `$HOME/OpenFOAM/blastfoam`

```
mkdir -p $HOME/OpenFOAM # create the OpenFOAM directory
cd $HOME/OpenFOAM # go to the directory
git clone https://github.com/synthetik-technologies/blastfoam
```

1.3 Optional download and compilation of OpenQBMM

In addition to the standard *blastFoam* solvers, additional solvers can be compiled using the *OpenQBMM* library, which has been developed to solve number density transport using the quadrature based method of moments.

Summary of steps to download, configure and compile *OpenQBMM*:

```
# 1. go to the $HOME/OpenFOAM directory
cd $HOME/OpenFOAM

# 2. clone the OpenQBMM repository
git clone https://github.com/OpenQBMM/OpenQBMM.git

# 3. Add the location of OpenQBMM to your $HOME/.bashrc file
echo "export QBMM_INST_DIR=$HOME/$WM_PROJECT/OpenQBMM" >> $HOME/.bashrc

# 4. go to the OpenQBMM directory
cd $HOME/OpenFOAM/OpenQBMM

# Compile OpenQBMM (for parallel use "-j")
./Allwmake
```

1.4 Compilation

After cloning the the *blastFoam* source code from https://github.com/synthetik-technologies/blastfoam (see previous section) and installing any optional libraries, similar to OpenFOAM, add the following line to `$HOME/.bashrc` file (must be after QBMM_INST_DIR):

```
source ~/OpenFOAM/blastfoam/etc/bashrc
```

and run:

```
source ~/.bashrc
```

in any open terminals to update the BASH environment. This is used to specify the local directories included during *blastFoam* compilation, as well as make future developments easier and less intrusive. The default path assumes that the *blastFoam* directory is located in `$HOME/OpenFOAM/blastfoam`. If this is not the case, the location will need to be changed in the `etc/bashrc` file by setting `BLAST_DIR` to the correct location.

Next run the `./Allwmake` command from the top `blastfoam` directory to compile the libraries and applications. Finally, ensure that OpenFOAM-7 and *blastFoam* have been installed and that the environment has been correctly setup by running one of the tutorial or validation cases.

Summary of steps to download, configure and compile *blastFoam*:

```
# 1. create the OpenFOAM directory
mkdir -p $HOME/OpenFOAM

# 2. go to the $HOME/OpenFOAM directory
cd $HOME/OpenFOAM

# 3. # clone the blastFoam repository
git clone https://github.com/synthetik-technologies/blastfoam

# go to the blastfoam directory
cd $HOME/OpenFOAM/blastfoam

# Append the etc/bashrc to your .bashrc file
echo "source $HOME/OpenFOAM/blastfoam/etc/bashrc" >> $HOME/.bashrc
```

```
# Load and set the bash environment to compile blastFoam
source $HOME/.bashrc

# Compile blastFoam (for parallel use "-j")
./Allwmake
```

1.5 Executable

The applications to solve these equations are executed by running the executable name (i.e. *blastFoam*). The executables are stored within the `$FOAM_USER_BIN` directory, and can be run from any directory.

1.6 Getting Help

This guide will attempt to cover the major points of the solver, but for more in depth question on the equations or models, please see the references listed in the class header files (*.H). Please report bugs using the issues tab on the GitHub page: https://github.com/synthetik-technologies/blastfoam/issues

2 Governing equations

Here the governing conservation equations used within *blastFoam* will be described. We will describe two cases: the single fluid case where one single, shared velocity and energy is used, and the multi-fluid case where each fluid has a unique velocity and energy. The phrase "multi-phase" will be used to denote a fluid that uses multiple equations of states to describe the density-energy-pressure relation. For both variations $\mathbf{U}$ is the vector of conservative variables, $\mathbf{F}$ are the fluxes corresponding to the respective conservative variables, and $\mathbf{S}$ is a vector of source terms,

$$\partial_t \mathbf{U} + \boldsymbol{\nabla} \cdot \mathbf{F} = \mathbf{S}. \tag{2.1}$$

The exact form each of these take will depend on where the single-fluid or multi-fluid is used, where both are described below.

2.1 Single-fluid model

For a single phase with multiple species, the conservative variables and fluxes are defined as

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho Y_s \\ \rho \mathbf{u} \\ \rho E \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \rho \mathbf{u} \\ \rho Y_s \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 0 \\ \dot{R}_s \\ \dot{\mathrm{M}} + \dot{\mathrm{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix}, \tag{2.2}$$

where Y_s is the mass fraction of specie s. And for multiple phases

$$\mathbf{U} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ \alpha_1\rho_1 \\ \vdots \\ \alpha_n\rho_n \\ \rho \mathbf{u} \\ \rho E \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \alpha_1 \mathbf{u} \\ \vdots \\ \alpha_n \mathbf{u} \\ \alpha_1\rho_1 \mathbf{u} \\ \vdots \\ \alpha_n\rho_n \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} \alpha_1 \boldsymbol{\nabla} \cdot \mathbf{u} \\ \vdots \\ \alpha_n \boldsymbol{\nabla} \cdot \mathbf{u} \\ 0 \\ \vdots \\ 0 \\ \dot{\mathbf{M}} + \dot{\mathbf{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix}, \tag{2.3}$$

where ρ is the mixture density, $\mathbf{u}$ the mixture velocity, E the total energy (thermal and kinetic), p the pressure, and ρ_i and α_i are the density and

volume fraction of each phase. The source terms, $\dot{R}_s$, $\dot{\mathrm{M}}$, and $\dot{\mathrm{E}}$ are the specie production from reactions, momentum, and energy sources (including gravitational acceleration), and $\dot{\mathrm{M}}_v$ and $\dot{\mathrm{E}}_v$ are viscous terms.

The sum of all volume fractions is defined to be one, i.e.

$$\sum_i \alpha_i = 1 \tag{2.4}$$

When only two phases are used, the second volume fraction is defined

$$\alpha_2 = 1 - \alpha_1. \tag{2.5}$$

Additionally, the mixture density is defined by

$$\rho = \sum_i \alpha_i \rho_i. \tag{2.6}$$

When all phases are considered inviscid, $\dot{\mathrm{M}}_v = \dot{\mathrm{E}}_v = 0$. However, when viscosity is included,

$$\dot{\mathbf{M}}_v = \boldsymbol{\nabla} \cdot \left(\rho\nu \left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^T - \frac{2}{3}(\boldsymbol{\nabla} \cdot \mathbf{u})\mathbf{I} \right] \right), \tag{2.7}$$

and

$$\dot{\mathrm{E}}_v = \boldsymbol{\nabla} \cdot \left(\rho\nu \left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^T - \frac{2}{3}(\boldsymbol{\nabla} \cdot \mathbf{u})\mathbf{I} \right] \cdot \mathbf{u} \right) + \boldsymbol{\nabla} \cdot (\alpha_T \boldsymbol{\nabla} e), \tag{2.8}$$

where in ν is the viscosity, α_T is the thermal diffusivity, and $\mathbf{I}$ is the identity matrix.

The five equation model is used for multiphase systems in which a single shared velocity, shared internal energy, and temperature is used, and transports individual phase masses and volume fractions. Pressure is defined using a specified EOS, where the mixture's internal energy, densities, and volume fraction are used to calculate the total pressure.

2.2 Multi-fluid

Similar to flows that use a single shared velocity and energy the mass, momentum, and energy are all conserved. However, in contrast, each phase

has its own set of mass, momentum and energy which is transferred between phases using interfacial transfer models (such as drag, heat transfer, etc.).

The conservation equations for the ith fluid phase are

$$\mathbf{U}_i = \begin{pmatrix} \alpha_i \\ \alpha_i \rho_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i E_i \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \alpha_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \otimes \mathbf{u}_i + p_i \mathbf{I} \\ \alpha_i (\rho_i E_i + p_i) \mathbf{u}_i \end{pmatrix}, \tag{2.9}$$

$$\mathbf{S} = \begin{pmatrix} \alpha_i \boldsymbol{\nabla} \cdot \mathbf{u}_i \\ 0 \\ \alpha_i \rho_i \mathbf{g} + \sum_{i \neq j} \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \alpha_i \rho_i \mathbf{g} \cdot \mathbf{u}_i + \sum_{i \neq j} (\dot{\mathrm{E}}_{ij} + \dot{\mathbf{M}}_{ij} \cdot \mathbf{u}_{int,ij}) + \dot{\mathrm{E}}_{det,i} + \dot{\mathrm{E}}_{v,i} \end{pmatrix}, \tag{2.10}$$

where α_i is the phase volume fraction, ρ_i is the phase density, $\mathbf{u}_i$ the phase velocity, E_i the total phase energy, and p_i the phase pressure. The source terms, $\dot{\mathbf{M}}_{ij}$ and $\dot{\mathrm{E}}_{ij}$ are the momentum and energy sources which account for interactions between other phases, $\dot{\mathrm{M}}_{v,i}$ and $\dot{\mathrm{E}}_{v,i}$ are viscous terms, $\mathbf{g}$ is the gravitational acceleration vector, $\dot{\mathrm{E}}_{det,i}$ is the energy added due to reactions, and $\mathbf{u}_{int,ij}$ is the interfacial velocity between two phases.

The viscous stress used a Newtonian stress tensor defined as

$$\dot{\mathbf{M}}_{v,i} = \boldsymbol{\nabla} \cdot \left(\alpha_i \rho_i \nu_i \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^T - \frac{2}{3} (\boldsymbol{\nabla} \cdot \mathbf{u}_i) \mathbf{I} \right] \right), \tag{2.11}$$

and

$$\dot{\mathrm{E}}_{v,i} = \boldsymbol{\nabla} \cdot \left(\alpha_i \rho_i \nu_i \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^T - \frac{2}{3} (\boldsymbol{\nabla} \cdot \mathbf{u}_i) \mathbf{I} \right] \cdot \mathbf{u}_i \right) + \boldsymbol{\nabla} \cdot (\alpha_i \alpha_{T,i} \boldsymbol{\nabla} e_i), \tag{2.12}$$

where in ν_i is the viscosity, $\alpha_{T,i}$ is the thermal diffusivity, and $\mathbf{I}$ is the identity matrix.

The momentum transfer term, $\dot{\mathbf{M}}_{ij}$, is defined as

$$\dot{\mathbf{M}}_{ij} = \dot{\mathbf{M}}_{D,ij} + \dot{\mathbf{M}}_{L,ij} + \dot{\mathbf{M}}_{VM,ij} + \dot{\mathbf{M}}_{WL,ij} + \dot{\mathbf{M}}_{TD,ij} \tag{2.13}$$

which are, in order, drag, lift, virtual mass, wall lubrication, and turbulent dispersion.

The energy transfer term, $\dot{\mathrm{E}}_{ij}$, is defined as

$$\dot{\mathrm{E}}_{ij} = \dot{\mathrm{E}}_{HT,ij} \tag{2.14}$$

where $\dot{\mathrm{E}}_{HT,ij}$ is the energy transfer rate due to convective heat transfer.

2.2.1 Granular phases

In contrast to a standard fluid (gas or liquid), a granular fluid tracks the phase mass, momentum, thermal energy, and granular energy. The last quantity, granular energy is the energy that is present due to fluctuation in the phase velocity within a cell. This is the kinetic energy due to the local velocity variance. The more common quantity is the granular temperature (Θ_i) which is related to the granular energy by $\langle E \rangle_i = \frac{3}{2}\Theta_i$. The governing equations are given below

$$\mathbf{U}_i = \begin{pmatrix} \alpha_i \rho_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i e_i \\ \alpha_i \rho_i \langle E \rangle_i \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \otimes \mathbf{u}_i + (P_{s,i} + P_{fric,i})\mathbf{I} \\ \alpha_i \rho_i e_i \mathbf{u}_i \\ \alpha_i \rho_i \langle E \rangle_i \end{pmatrix}, \tag{2.15}$$

$$\mathbf{S} = \begin{pmatrix} 0 \\ \alpha_i \rho_i \mathbf{g} - \alpha_i \sum_l \boldsymbol{\nabla} p_{int,il} + \sum_j \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \sum_j \dot{\mathrm{e}}_{ij} + \sum_k \dot{\gamma}_{ik} + \dot{e}_{det,i} \\ -\sum_k \dot{\gamma}_{ik} - \sum_l \phi_{prod,il} - P_{s,i} \boldsymbol{\nabla} \cdot \mathbf{u}_i + \alpha_i \boldsymbol{\tau}_i : \boldsymbol{\nabla} \mathbf{u}_i + \boldsymbol{\nabla} \cdot (\kappa_{s,i} \boldsymbol{\nabla} \Theta_i) \end{pmatrix}, \tag{2.16}$$

where again α_i is the phase volume fraction, ρ_i is the phase density, $\mathbf{u}_i$ is the phase velocity, and e_i is the phase thermal energy. The summations $\sum_j$, $\sum_k$, and $\sum_l$ are taken to mean the summation over all phases, all solid phases, and all fluid phases. The new quantities, $P_{s,i}$, $P_{fr,i}$, $\langle E \rangle$, and p_{int} are the granular pressure, frictional pressure, granular energy, and interfacial pressure (generally taken as the gas phase pressure) respectively. The density is assumed to be constant, so the volume fraction is found by simply dividing the total phase mass by the known density. The source terms, $\dot{\mathbf{M}}_{ij}$, $\dot{\mathrm{e}}_{ij}$, $\dot{\gamma}_{ij}$, $\phi_{prod,ij}$ and are the momentum transfer, energy transfer, and granular energy dissipation and production terms. The momentum and energy transfer terms are the same as for a fluid phase ($\dot{\mathrm{e}}_{ij} = \dot{\mathrm{E}}_{ij}$ and $\dot{\mathrm{e}}_{det,i} = \dot{\mathrm{E}}_{det,i}$), and the production and dissipation terms are described below.

The production and dissipation of granular energy introduce a new term into the continuous phase total energy equation so that the continuous phase

source terms are now

$$\mathbf{S} = \begin{pmatrix} \alpha_i \boldsymbol{\nabla} \cdot \mathbf{u}_i \\ 0 \\ \alpha_i \rho_i \mathbf{g} + \sum_{i \neq j} \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \alpha_i \rho_i \mathbf{g} \cdot \mathbf{u}_i + \sum_{i \neq j} \dot{\mathrm{E}}_{ij} + \dot{\mathrm{E}}_{det,i} + \dot{\mathrm{E}}_{v,i} + \sum_k \phi_{prod,k} \end{pmatrix}. \tag{2.17}$$

You can also see in Eq. (2.15) that the dissipation of granular energy is converted into thermal energy within the solid phase. For those familiar with incompressible flows, this term is typically neglected due to the fact that the amount of heat produced from collisions is typically very small. In highly compressible flows, this is not the case, and significant heat can occur due to the loss of energy from inelastic collisions.

We define the change in momentum due to granular phase stress as

$$\dot{\mathbf{M}}_{ij} = \boldsymbol{\nabla} \cdot \boldsymbol{\tau}_{s,i}, \tag{2.18}$$

and the granular stress is defined as

$$\boldsymbol{\tau}_{s,i} = \rho_i \nu_{s,i} \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^T \right] + (\lambda_{s,i} - \frac{2}{3} \rho_i \nu_{s,i})(\boldsymbol{\nabla} \cdot \mathbf{u}_i)\mathbf{I}, \tag{2.19}$$

with $\lambda_{s,i}$ being the bulk viscosity of the granular phase defined by (Huilin and Gidaspow, 2003)

$$\lambda_{s,i} = \sum_k P_{s,ik} \frac{d_{ik}}{3} \sqrt{\frac{2(m_i \Theta_i + m_k \Theta_k)^2}{\pi \Theta_i \Theta_k (m_i^2 \Theta_i + m_k^2 \Theta_k)}}, \tag{2.20}$$

where $P_{s,ij}$ is the granular pressure between solid phases i and k, and is related to $P_{s,i}$ by

$$P_{s,i} = \alpha_i \rho_i \Theta_i + \sum_k P_{s,ik}. \tag{2.21}$$

The production of granular energy is given by

$$\phi_{prod,il} = \frac{81 \alpha_i \mu_l^2}{g_{0,ii} d_i^3 \rho_i \sqrt{\pi}} \frac{|\mathbf{u}_l - \mathbf{u}_i|^2}{\sqrt{\Theta_i}} - 3 K_{D,il} \Theta_i, \tag{2.22}$$

where $K_{D,il}$ is the drag coefficient for the i-l solid-fluid pair and $g_{0,ij}$ is the radial distribution function.

The dissipation of granular energy between granular phases i and k is defined by (Huilin and Gidaspow, 2003)

$$\dot{\gamma}_{ik} = \left[\frac{3}{d_{ik}} \left(\frac{2m_0^2 \Theta_i \Theta_k}{\pi(m_i^2 \Theta_i + m_k^2 \Theta_k)} \right)^{1/2} - \frac{3m_0(m_i \Theta_i + m_k \Theta_k)}{4(m_i^2 \Theta_i + m_k^2 \Theta_k)} \boldsymbol{\nabla} \cdot \mathbf{u}_i \right] (1 - e_{ik}) P_{s,ik}, \tag{2.23}$$

where

$$m_i = \frac{\rho_i \pi d_i^3}{6}, \tag{2.24}$$

$d_{ik} = \frac{d_i + d_k}{2}$, $m_0 = m_i + m_k$, e_{ik} is the coefficient of restitution between granular phases i and j, and $P_{s,ik}$ is the granular pressure between phases i and j. The drag, radial distribution function, and granular pressure are determined by models and will be discussed in Sec. 2.3.1 and Sec. 2.4. The specific manner as to how these source terms are solved can be found in Appendix B.7 of Houim and Oran (2016).
Lastly, the speed of sound for a granular material is given by

$$c_i^2 = \frac{1}{\rho_i} \left[\frac{\partial P_{tot,i}}{\partial \alpha_i} + \frac{2}{3} \frac{\Theta_i \left(\frac{\partial P_{tot,i}}{\partial \Theta_i} \right)^2}{\rho_i \alpha_i^2} \right] \tag{2.25}$$

where $P_{tot,i} = P_{s,i} + P_{fr,i}$. Because different models can be used for the granular pressure, frictional pressure, and additional models used within (radial distribution functions), the specific form this takes will not be discussed.

The topic of granular flows, or kinetic theory, is extensive and the derivations and rationals for the equations and models presented are extremely math intensive and well outside the scope of this user guide. However for those interested in the math behind these equations and models it is recommend that the reader sees the following sources: Chapman et al. (1990) for a very mathematical derivation of the governing equations, Gidaspow (1994) for more practical derivations, Fox (2019) for the application to compressible flows, and Huilin and Gidaspow (2003); Chao et al. (2011) for derivations of the polydisperse formulations.

2.3 Interfacial models

The exchange of momentum and energy is determined by the use of interfacial models. These models are described below, and include the equations used to calculate the transfer rates. In general these models are solved using an explicit Euler integration method, however drag and heat transfer are solved using an analytical solution and the reference used for these solution methods will be provided in the relevant sections.

NOTE: There are more models currently implemented, however they do not apply to gas-particle flows and will not be discussed. Their descriptions and equations will be added as they become relevant.

2.3.1 Drag models

The drag is responsible for relaxing the velocity difference between two phases to an equilibrium velocity at a rate determined by the drag coefficient, $K_{D,ij}$. The momentum transfer term is defined as $\dot{\mathbf{M}}_{D,ij} = K_{D,ij}(\mathbf{u}_j - \mathbf{u}_i)$. The subscripts c and d are used to denote the continuous and dispersed phases respectively. The solution to the drag relaxation can be found in Appendix B.7 of Houim and Oran (2016). Additionally, an ODE solver can be used to solve the drag relaxation, however this is approach is much more computationally expensive.

$$K_{D,ij} = 0.75\frac{\alpha_d C_D Re_{ij} \rho_c \nu_c}{d_d^2}, \tag{2.26}$$

where the Reynolds number between phases i and j are defined as $Re_{ij} = \frac{d_d|\mathbf{u}_c - \mathbf{u}_d|}{\nu_c}$. The term $C_D Re_{ij}$ will be defined by the models below.

Variable	Description
swarmCorrelation	Swarm model

2.3.1.1 Schiller Naumann

The most basic drag model was derived by Schiller and Naumann (1933) and calculates the drag coefficient using flow over a single sphere

$$C_D Re_{ij} = \begin{cases} 24(1 + 0.15 Re_{ij}^{0.687}) & \text{if } Re_{ij} < 1000 \\ 0.44 Re_{ij} & \text{else} \end{cases}. \tag{2.27}$$

Variable	Description
$residualRe$	Minimum Reynolds number [ ]

2.3.1.2 Wen Yu

The Wen and Yu drag model (Enwald et al., 1996) is used for dispersed particulate flow and the drag coefficient is defined as

$$C_D Re_{ij} = C_D Res_{SN,ij} \alpha_c^{-2.65}, \tag{2.28}$$

where $C_D Re_{SN,ij}$ is the Schiller Naumman drag coefficient multiplied by the Reynolds number, and the Reynolds number used in the drag coefficient is weighted by the dispersed phase volume fraction, i.e.

$$Res_{ij} = Re_{ij} \alpha_c. \tag{2.29}$$

Variable	Description
$residualRe$	Minimum Reynolds number [ ]

2.3.1.3 Ergun

The Ergun drag model (Enwald et al., 1996) is used for dispersed particulate flow and the drag coefficient is defined as

$$C_D Re_{ij} = \frac{4}{3} \frac{150(1-\alpha_c)}{\alpha_c} + 1.75 Re_{ij} \tag{2.30}$$

2.3.1.4 Gibilaro

The Gibilaro drag model (Enwald et al., 1996) is used for dispersed particulate flow and the drag coefficient is defined as

$$C_D Re_{ij} = \frac{4}{3} (\frac{17.3}{\alpha_2} + 0.336 Re_{ij}) \alpha_c \alpha_2^{-2.8} \tag{2.31}$$

and $\alpha_2 = 1 - \alpha_d$.

2.3.1.5 Syamlal O'Brien

The Syamlal O'Brien drag model (Syamlal et al., 1993) is used for dispersed particulate flow and the drag coefficient is defined as

$$C_D Re_{ij} = \frac{C_{Ds} Re_{ij} \alpha_c}{V_r^2}, \tag{2.32}$$

where

$$C_{Ds} Re_{ij} = \left(0.63\sqrt{Re_{ij}} + 4.8\sqrt{V_r}\right)^2, \tag{2.33}$$

$$V_r = 0.5\left(A - 0.06 Re_{ij} + \sqrt{(0.06 Re_{ij})^2 + 0.12 \Re_{ij}(2B - A) + A^2}\right), \tag{2.34}$$

$$A = \alpha_2^{4.14}, \tag{2.35}$$

$$B = \begin{cases} 0.8\alpha_2^{1.28} & \text{if } \alpha_2 < 0.85 \\ \alpha_2^{2.65} & \text{else} \end{cases}, \tag{2.36}$$

and $\alpha_2 = 1 - \alpha_d$.

2.3.1.6 Gidaspow-Ergun-Wen Yu

The Gidaspow, Wen and Yu drag model (Gidaspow, 1994) is used for dispersed particulate flow and the drag coefficient is a combination of the Wen and Yu drag model (dilute) and Ergun drag model (dense)

$$C_D Re_{ij} = \begin{cases} C_D Re_{WenYu,ij} & \text{if } \alpha_c > 0.80 \\ C_D Re_{Ergun,ij} & \text{else} \end{cases} \tag{2.37}$$

where $C_D Re_{WenYu,ij}$ is the Wen Yu drag coefficient and $C_D Re_{Ergun,ij}$ is the Ergun drag coefficient, both multiplied by the Reynolds number.

Variable	Description
$residualRe$	Minimum Reynolds number [ ]

2.3.1.7 Gidaspow-Schiller Naumann

The Gidaspow-Schiller Naumann drag model (Enwald et al., 1996) is used for dispersed particulate flow and the drag coefficient is defined as

$$C_D Re_{ij} = C_{Ds} Re_{ij} \alpha_c^{-1.65}, \tag{2.38}$$

$$C_{Ds} Re_{ij} = \begin{cases} \frac{24(1+0.15 Res_{ij}^{0.687})}{\alpha_c} & \text{if } Res_{ij} < 1000 \\ 0.44 Res_{ij} & \text{else} \end{cases} \tag{2.39}$$

where the Reynolds number used in the drag coefficient is weighted by the dispersed phase volume fraction, i.e.

$$Res_{ij} = Re_{ij} \alpha_c. \tag{2.40}$$

Variable	Description
$residualRe$	Minimum Reynolds number [ ]

2.3.2 Lift models

The lift interfacial momentum transfer model is used to account for acceleration due to rotation in the continuous phase and is defined as

$$\dot{\mathbf{M}}_{L,ij} = C_{L,ij} \alpha_d \rho_c \left[(\mathbf{u}_c - \mathbf{u}_i) \times (\boldsymbol{\nabla} \times \mathbf{u}_c) \right] \tag{2.41}$$

where $C_{L,ij}$ is the lift coefficient between phases i and j, and is defined by the model of choice

2.3.2.1 Constant coefficient

The lift coefficient is a constant value.

Variable	Description
Cl	Lift coefficient [ ]

2.3.3 Heat transfer models

The heat transfer model is responsible for relaxing the temperature between two phase at a rate determined by the model of choice. The Prandtl number will be used and is defined as

$$Pr_{ij} = \frac{\nu_c C_{vc} \rho_c}{\kappa_c}. \tag{2.42}$$

Again, the subscripts c and d represent the continuous and dispersed phases respectively. C_{vc} is the continuous phase specific heat at constant volume and κ_c is the continuous phase thermal conductivity. The solution to the temperature relaxation can be found in Appendix B.7 of Houim and Oran (2016).

2.3.3.1 Ranz Marshall

The Ranz Marshall heat transfer model (Houim and Oran, 2016) defines the heat transfer rate as

$$\dot{\mathrm{E}}_{HT,ij} = \frac{6\alpha_d \kappa_c N u_{ij}}{d_d^2}, \tag{2.43}$$

where the Nusselt number is defined as

$$\begin{aligned} Nu_{ij} =&(7 - 10\alpha_c + 5.0\alpha_c^2)(1 + 0.7Re_{ij}^{0.2} Pr_{ij}^{1/3}) \\ &+(1.33 - 2.4\alpha_c + 1.2\alpha_c^2) Re_{ij}^{0}.7 Pr_{ij}^{1/3}. \end{aligned} \tag{2.44}$$

2.3.3.2 Constant Nusselt number

For the constant Nusselt number heat transfer model, Eq. (2.43) is again used, but the Nusselt number is a constant value provided by the user.

Variable	Description
Nu	Nusselt number [ ]

2.4 Granular phase models

For granular phases, additional models are required to account for particle-particle collisions in both the kinetic and frictional regimes. These models include radial distribution, viscosity, conductivity, granular pressure, frictional

pressure, solid-solid drag, and packing models. Below are some common variables that will be used in the granular models
$\sum_k$ is the summation over all granular phase indices and α_s is the sum of all granular phase volume fractions. The strain is defined as

$$\mathbb{S} = \frac{1}{2}\left[(\boldsymbol{\nabla}\mathbf{u})^T + \boldsymbol{\nabla}\mathbf{u}\right] - \frac{1}{3}\boldsymbol{\nabla}\cdot\mathbf{u}\mathbb{I}, \tag{2.45}$$

the symmetric strain is defined as

$$\mathbb{D} = \frac{1}{2}\left[(\boldsymbol{\nabla}\mathbf{u})^T + \boldsymbol{\nabla}\mathbf{u}\right]. \tag{2.46}$$

and several models use the total granular drag on a phase, i.e.

$$K_{D,i} = \sum_k K_{D,ik}. \tag{2.47}$$

2.4.1 Radial distribution models

The radial distribution gives the likely hood of a collision between two particles. This is defined between like and unlike particles phases.

2.4.1.1 Carnahan Starling

The Carnahan Starling radial distribution function is only valid for monodisperse flows and is defined as

$$g_{0,ii} = \frac{1}{1-\alpha_i} + \frac{3\alpha_i}{2(1-\alpha_i)^2} + \frac{\alpha_i^2}{2(1-\alpha_i)^3}. \tag{2.48}$$

2.4.1.2 Sinclair Jackson

The Sinclair Jackson radial distribution (Sinclair and Jackson, 1989) function is only valid for monodisperse flows and is defined as

$$g_{0,ii} = \frac{1}{1-\left(\frac{\alpha_i}{\alpha_{i,max}}\right)^{1/3}} \tag{2.49}$$

2.4.1.3 Lun Savage

The Lun Savage radial distribution function (Lun et al., 1984) is only valid for monodisperse flows and is defined as

$$g_{0,ii} = \left(1 - \frac{\alpha_i}{\alpha_{max,i}}\right)^{-2.5\alpha_{max,i}} \tag{2.50}$$

2.4.1.4 Gao

The Gao radial distribution function Gao et al. (2008) is a modified version of the Sinclair Jackson radial distribution function and is valid for polydisperse flows, and is defined as

$$g_{0,ij} = \frac{d_i g_{0,i} + d_j g_{0,j}}{d_i + d_j} \tag{2.51}$$

$$g_{0,i} = \frac{1}{1 - \left(\frac{\alpha_i}{\alpha_{i,max}}\right)^{1/3}} \tag{2.52}$$

2.4.1.5 Lebowitz

The Lebowitz radial distribution function Benyahia et al. (2012) is valid for polydisperse flows, and is defined as

$$g_{0,ij} = \frac{1}{1-\alpha_s} + \frac{3 d_i d_j}{(1-\alpha_s)^2 (d_i + d_j)} \sum_k \frac{\alpha_k}{d_k} \tag{2.53}$$

2.4.2 Granular viscosity models

The modeling of granular viscosity is used to determine the stress seen by a granular phases. These models only account for single collisions (frictional models will have additional contributions for mutliple collisions)

2.4.2.1 Gidaspow

The Gidaspow viscosity model (Gidaspow, 1994) is for monodisperse flows and is defined as

$$\begin{aligned} \nu_{s,i} =& \frac{4}{5}\alpha_i^2 g_{0,ii} \frac{(1+e_{i,j})}{\pi} + \frac{1}{15}\sqrt{\pi} g_{0,ii}(1+e_{ii})\alpha_i^2 \\ &+ \frac{1}{6}\sqrt{\pi}\alpha_i + \frac{10}{96}\frac{\sqrt{\pi}}{(1+e_{ii})g_{0,ii}} \end{aligned} \tag{2.54}$$

2.4.2.2 Hrenya Sinclair

The Hrenya Sinclair viscosity model (Hrenya and Sinclair, 1997) is for monodisperse flows and is defined as

$$\begin{aligned}\nu_{s,i} =& \frac{4}{5}\alpha_i^2 g_{0,ii}\frac{(1+e_{i,j})}{\pi} + \frac{1}{15}\sqrt{\pi}g_{0,ii}(3e_{ii}-1)\frac{\alpha_i^2}{3-e_{ii}} \\ &+\frac{1}{6}\frac{\sqrt{\pi}\alpha_i(0.5\lambda+0.25(3e_{ii}-1))}{0.5(3-e_{ii})\lambda} + \frac{10}{96}\frac{\sqrt{\pi}}{(1+e_{ii})0.5(3-e_{ii})g_{0,ii}\lambda}\end{aligned}, \quad (2.55)$$

with

$$\lambda = 1 + \frac{d_i L}{6\sqrt{2}\alpha_i}. \quad (2.56)$$

Variable	Description
L	characteristic length [m]

2.4.2.3 Princeton

The Princeton viscosity model (Agrawal et al., 2001) is for polydisperse flows and is defined as

$$\nu_{s,i} = \rho_i \frac{2+\alpha}{3}\left[\frac{\mu_i^*}{g_{0,ii}\eta(2-\eta)}\left(1+\frac{8}{5}\eta\sum_k(\alpha_k g_{0,ik})\right)\left(1+\frac{8}{5}\eta(3\eta-2)\sum_k(\alpha_k g_{0i,k})\right)+\frac{3}{5}\eta\mu_b\right], \quad (2.57)$$

with

$$\mu_i^* = \frac{\rho_i\alpha_i g_{0,ii}\Theta_i\mu}{\rho_i\sum_k(\alpha_k g_{0,ik})\Theta_i + \frac{2K_{d,i\mu}}{\rho_i\alpha_i}}, \quad (2.58)$$

$$\mu = \frac{5}{96}\rho_i d_i\sqrt{\pi\Theta_i}, \quad (2.59)$$

and

$$\mu_b\frac{256}{5\pi}\mu\alpha_i\sum_k(\alpha_k g_{0,ik}). \quad (2.60)$$

2.4.2.4 Syamlal

The Syamlal viscosity model (Syamlal et al., 1993) is for monodisperse flows and is defined as

$$\begin{aligned}\nu_{s,i} =& \frac{4}{5}\alpha_i^2 g_{0,ii}\frac{(1+e_{ii})}{\pi} + \frac{1}{15}\sqrt{\pi}g_{0,ii}(3e_{ii}-1)\frac{\alpha_i^2}{3-e_{ii}} \\ &+\frac{1}{6}\frac{\sqrt{\pi}\alpha_i}{(3-e_{ii})}\end{aligned} \tag{2.61}$$

2.4.2.5 None

No granular viscosity is used. This also removes the conductivity contribution to the granular energy equation.

2.4.3 Granular conductivity models

2.4.3.1 Chao

The Chao granular conductivity model Chao et al. (2011) is defined as

$$\kappa_{s,i} = \frac{2\kappa_{dilute,i}}{1/N\sum_k(1+e_{ik})g_{0,ik}}\left[1+\frac{6}{5}\sum_k \alpha_k(1+e_{ik})g_{0,ik}\right]^2, \tag{2.62}$$

where N is the number of granular phases and

$$\kappa_{dilute,i} = \frac{15\rho_i\Theta_i}{\sum_k\left[\pi\sqrt{2\pi}n_k d_{ij}^2(\sqrt{\Theta_i}+\sqrt{\Theta_j}-0.56\sqrt[4]{\Theta_i\Theta_j})\right]}. \tag{2.63}$$

2.4.3.2 Gidaspow

The Gidaspow conductivity model (Gidaspow, 1994) is for monodisperse flows and is defined as

$$\begin{aligned}\kappa_{s,i} =& 2\alpha_i^2 g_{0,ii}\frac{(1+e_{ii})}{\sqrt{\pi}} + \frac{9}{8}\sqrt{\pi}g_{0,ii}0.5(1+e_{ii})\alpha_i^2 \\ &+\frac{15}{16}\sqrt{\pi}\alpha_i + \frac{25}{64}\frac{\sqrt{\pi}}{g_{0,ii}(1+e_{ii})}.\end{aligned} \tag{2.64}$$

2.4.3.3 Hrenya Sinclair

The Hrenya Sinclair conductivity model (Hrenya and Sinclair, 1997) is for monodisperse flows and is defined as

$$\begin{aligned}\kappa_{s,i} =& 2\alpha_i^2 g_{0,ii}\frac{(1+e_{ii})}{\sqrt{\pi}} + \frac{9}{8}\frac{\sqrt{\pi}g_{0,ii}0.25(1+e_{ii})^2(2e_{ii}-1)\alpha_i^2}{\frac{49}{16}-\frac{33}{16}e_{ii}} \\ &+\frac{15}{16}\frac{\sqrt{\pi}\alpha_i(0.5e_{ii}^2+0.25e_{ii}-0.75+\lambda)}{\left(\frac{49}{16}-\frac{33}{16}e_{ii}\right)\lambda} \\ &+\frac{25}{64}\frac{\sqrt{\pi}}{g_{0,ii}\lambda(1+e_{ii})\left(\frac{49}{16}-\frac{33}{16}e_{ii}\right)}\end{aligned}, \tag{2.65}$$

with

$$\lambda = 1 + \frac{d_i L}{6\sqrt{2}\alpha_i}. \tag{2.66}$$

Variable	Description
L	characteristic length [m]

2.4.3.4 Princeton

The Princeton conductivity model (Agrawal et al., 2001) is for polydisperse flows and is defined as

$$\kappa_{s,i} = \frac{\kappa_i^*}{g_{0,ii}}\left[\begin{array}{c}\left(1+\frac{12}{5}\eta\sum_k(\alpha_k g_{0,ik})\right)\left(1+\frac{12}{5}\eta^2(4\eta-3)\sum_k(\alpha_k g_{0,ik})\right) \\ +\frac{64}{25\pi}(41-33\eta)\eta^2\left(\sum_k(\alpha_k g_{0,ik})\right)^2\end{array}\right], \tag{2.67}$$

with

$$\kappa_i^* = \frac{\rho_i\alpha_i g_{0,ii}\Theta_i\kappa_i}{\rho_i\sum_k(\alpha_k g_{0,ik})\Theta_i + \frac{6K_{d,i}\kappa_i}{5\rho_i\alpha_i}}, \tag{2.68}$$

and

$$\kappa_i = \frac{75\rho_i d_i\sqrt{\pi\Theta_i}}{48\eta(41-33\eta)}. \tag{2.69}$$

2.4.3.5 Syamlal

The Syamlal conductivity model (Syamlal et al., 1993) is for monodisperse flows and is defined as

$$\begin{aligned}\kappa_{s,i} =& 2\alpha_i^2 g_{0,ii}\frac{(1+e_{ii})}{\sqrt{\pi}} + \frac{9}{8}\frac{\sqrt{\pi}g_{0,ii}0.25(1+e_{ii})(2e_{ii}-1)\alpha_i^2}{\frac{49}{16}-3316e_{ii}} \\ &+\frac{15}{16}\frac{\sqrt{\pi}\alpha_i}{\frac{49}{16}-\frac{33}{16}e_{ii}}.\end{aligned} \tag{2.70}$$

2.4.4 Granular pressure models

The granular pressure model is used to calculate the pressure due to single collisions between particles. Pressures that occur due to multiple collisions will be added using the frictional models.

2.4.4.1 Huilin

The Huilin granular pressure model (Huilin and Gidaspow, 2003) is for polydisperse flows and is define as

$$\begin{aligned}P_{s,ij} =& \frac{\pi(1+e_{ij})d_{ij}^3 g_{0,ij}n_i n_j m_i m_j m_0 \Theta_i \Theta_j}{3(m_i^2\Theta_i + m_j^2\Theta_j)} \\ & \left[\frac{m_0^2\Theta_i\Theta_j}{(m_i^2\Theta_i + m_j^2\Theta_j)(\Theta_i+\Theta_j)}\right]^{3/2} \quad , \\ & (1-3\omega+6\omega^2-10\omega^3)\end{aligned} \tag{2.71}$$

with

$$\omega = \frac{m_i\Theta_i - m_j\Theta_j)}{[(m_i^2\Theta_i^2 + m_j^2\Theta_j^2) + \Theta_i\Theta_j(m_i^2+m_j^2)]^{1/2}}. \tag{2.72}$$

2.4.4.2 Lun

The Lun granular pressure model (Lun et al., 1984) is for monodisperse flows and is defined as

$$P_{s,ij} = \alpha_i\alpha_j\rho_i\Theta_i 2(1+e_{ij})g_{0,ij}. \tag{2.73}$$

2.4.5 Frictional stress models

The frictional stress model acts as a simple way to account for non-singular collisions between particles and results in very large stress when the granular phases approach their packing limit. This results in both a modeled pressure and viscosity to calculate a stress, and limits the ability of particles to move between computational cells. The frictional pressure and viscosity denoted P_{fr} and ν_{fr} act on all granular phases since the packing of particles is dependent on the total volume fraction of particles.

2.4.5.1 Johnson Jackson

The Johnson-Jackson frictional stress model (OpenCFD Ltd., 2018b) uses a smooth function to calculate the pressure

$$P_{fr} = \begin{cases} Fr\frac{(\alpha_s - \alpha_{crit,s})^\eta}{(\alpha_{max,s} - \alpha_s)^p} & \text{if } \alpha_s > \alpha_{crit,s} \\ 0 & \text{else,} \end{cases} \tag{2.74}$$

and the frictional viscosity is

$$\nu_{fr} = 0.5 P_{fr} \sin(\phi), \tag{2.75}$$

where ϕ is the angle of internal friction, and Fr, η, and p are model coefficients generally taking values of 0.05, 2, and 5 respectively. $\alpha_{crit,s}$ is the granular particle fraction that the frictional pressure "turns on", and $\alpha_{max,s}$ is the maximum packing limit of all granular phases. Generally $\alpha_{crit,s}$ is taken to be around 0.5.

Variable	Description
Fr	Model coefficient [ ]
η	Model coefficient [ ]
p	Model coefficient [ ]
ϕ	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{max,s}$ [ ]
$\alpha_{crit,s}$ (alphaMinFriction)	Volume fraction friction is "turned on" [ ]

2.4.5.2 Schaeffer

The Scheffer frictional stress model (Syamlal et al., 1993) uses a power law and the local stress is used to calculate the frictional viscosity. The critical

volume fraction, $\alpha_{crit,s}$ should be a value close to that of the packing limit. Because the local packing limit can vary in space when polydispersity is present, the ration of the local volume fraction by the critical volume fraction is used, i.e. $\alpha_{crit,s} = \left(\frac{\alpha_{crit,s}}{\alpha_{max,s}}\right)_{crit} \alpha_{max,s}$ where $\left(\frac{\alpha_{crit,s}}{\alpha_{max,s}}\right)_{crit}$ is specified by the user and $\alpha_{max,s}$ is the calculated local packing limit. The pressure is defined as

$$P_{fr} = \begin{cases} 10^{24}(\alpha_s - \alpha_{crit,s})^{10} & \text{if } \alpha_s > \alpha_{crit,s} \\ 0 & \text{else} \end{cases}, \tag{2.76}$$

and the viscosity is

$$\nu_{fr} = \begin{cases} 0.5\frac{P_{fr}\sin(\phi)}{\sqrt{\frac{1}{3}(I_1(\mathbb{D})^2 - I_2(\mathbb{D}))}} & \text{if } \alpha_s > \alpha_{crit,s} \\ 0 & \text{else} \end{cases}, \tag{2.77}$$

where $I_1(\mathbb{D})$ is the first in variant (or trace) and $I_2(\mathbb{D})$ is the second invariant.

Variable	Description
ϕ	Model coefficient [ ]
$\left(\frac{\alpha_{crit,s}}{\alpha_{max,s}}\right)_{crit}$ (alphaMinFrictionByAlphap)	Volume fraction ratio friction is "turned on" [ ]

2.4.5.3 Johnson Jackson-Schaeffer

For the Johnson Jackson-Scheffer frictional stress model (OpenCFD Ltd., 2018b), the frictional pressure of Johnson Jackson is used while the viscosity of the Schaeffer model is used.

$$P_{fr} = \begin{cases} Fr\frac{(\alpha_s - \alpha_{crit,s})^{\eta}}{(\alpha_{max,s} - \alpha_s)^{p}} & \text{if } \alpha_s > \alpha_{crit,s} \\ 0 & \text{else} \end{cases} \tag{2.78}$$

$$\nu_{fr} = \begin{cases} 0.5\frac{P_{fr}\sin(\phi)}{\sqrt{\frac{1}{3}(I_1(\mathbb{D})^2 - I_2(\mathbb{D}))}} & \text{if } \alpha_s > \alpha_{crit,s} \\ 0 & \text{else} \end{cases} \tag{2.79}$$

Variable	Description
Fr	Model coefficient [ ]
η	Model coefficient [ ]
p	Model coefficient [ ]
ϕ	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{max,s}$ [ ]
$\alpha_{crit,s}$ (alphaMinFriction)	Volume fraction friction is "turned on" [ ]

2.4.5.4 Princeton

The Princeton model Agrawal et al. (2001) uses a blending of the Johnson Jackson model when the local volume fraction is greater than $\alpha_{crit,JJ,s}$ but less than $\alpha_{crit,Sc,s}$. As with the Schaeffer model, the critical ratio of $\left(\frac{\alpha_{crit,Sc,s}}{\alpha_{max,s}}\right)_{crit}$ is used as an input rather that $\alpha_{crit,Sc,s}$.

$$P_{fr} = \begin{cases} 10^{24}(\alpha_s - \alpha_{crit,Sc,s})^{10} & \text{if } \alpha_s > \alpha_{crit,Sc,s} \\ Fr\frac{(\alpha_s - \alpha_{crit,JJ,s})^{\eta}}{(\alpha_{max,s} - \alpha_s)^{p}} & \text{if } \alpha_{crit,Sc,s} \leq \alpha_s > \alpha_{crit,JJ,s} \\ 0 & \text{else.} \end{cases} \tag{2.80}$$

The frictional viscosity is defined using

$$\frac{P_{c,k}}{P_{fr}} = \left(1 - \frac{\boldsymbol{\nabla} \cdot \mathbf{u}_s}{n\sqrt{2}\sin(\phi)\sqrt{\mathbb{S} : \mathbb{S} + \Theta_k / d_k^2}}\right)^{n-1}, \tag{2.81}$$

$$\nu_{fr,k} = \frac{\sqrt{2}P_{c,k}\sin(\phi)}{\sqrt{\mathbb{S} : \mathbb{S} + \Theta_k / d_k^2}} \left[n - (n-1)\left(\frac{P_{c,k}}{P_{fr}}\right)^{\frac{1}{1-n}}\right] \frac{\alpha_k}{\alpha_s}, \tag{2.82}$$

and

$$n = \begin{cases} \frac{\sqrt{3}}{2\sin(\phi)} & \text{if } \boldsymbol{\nabla} \cdot \mathbf{u}_s \geq 0 \\ 1.03 & \text{else} \end{cases}. \tag{2.83}$$

Variable	Description
Fr	Model coefficient [ ]
η	Model coefficient [ ]
p	Model coefficient [ ]
ϕ	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{max,s}$ [ ]
$\alpha_{crit,JJ,s}$ (alphaMinFriction)	Volume fraction friction is "turned on" [ ]
$\left(\frac{\alpha_{crit,Sc,s}}{\alpha_{max,s}}\right)_{crit,Sc,s}$ (alphaMinFrictionByAlphap)	Volume fraction ratio friction is "turned on" [ ]

2.4.6 Packing limit models

The modeling of the packing limit is important in polydisperse flows due to the fact that when large variation in particle diameters exist locally the packing limit can vary widely. This is due to the fact that smaller particles can be found in between larger particles, increasing the maximum allowable volume fraction. For monodisperse flows, this is not the case and a constant value packing limit should be used.

2.4.6.1 Constant packing limit

A constant value for the packing limit is used.

Variable	Description
alphaMax	Packing limit [ ]

2.4.6.2 Yu Standish packing limit

The Yu Standish packing model (Yu and Standish, 1987) assumes that the particle phases are ordered from smallest diameter to largest for the calculation of the packing limit. The packing limit is defined as

$$\alpha_{max,s} = \min \left[\frac{\alpha_{max,i}}{1 - \sum_{k=1}^{i-1} \left(1 - \frac{\alpha_{max,i}}{p_{ik}}\right) \frac{cx_i}{X_{ik}} - \sum_{k=i+1}^{N} \left(1 - \frac{\alpha_{max,i}}{p_{ik}}\right) \frac{cx_i}{X_{ik}}} \right] \quad \text{i = 1,2,...,N,} \tag{2.84}$$

where

$$cx_i = \frac{\alpha_i}{\alpha_s}, \tag{2.85}$$

$$X_{ik} = \begin{cases} \frac{1-r_{ik}^2}{2-\alpha_{max,i}} & \text{if } k < i \\ 1 - \frac{1-r_{ik}^2}{2-\alpha_{max,i}} & \text{if } k \geq i \end{cases}, \tag{2.86}$$

$$p_{ik} = \begin{cases} \alpha_{max,i} + \alpha_{max,i}(1-\alpha_{max,i})(1 - 2.35 r_{ik} + 1.35 r_{ik}^2) & \text{if } r_{ik} \leq 0.741 \\ \alpha_{max,i} & \text{if } r_{ik} > 0.741, \end{cases} \tag{2.87}$$

and

$$r_{ik} = \begin{cases} \frac{d_i}{d_k} & \text{if } i \geq k \\ \frac{d_k}{d_i} & \text{if } i < k \end{cases}. \tag{2.88}$$

Variable	Description
residualAlpha	Minimum volume fraction to divide by [ ]

2.4.6.3 Fedors-Landel packing limit

The Fedors-Landel packing model (Fedors and Landel, 1979) is only valid of for a binary mixture and is defined as

$$\alpha_{max,s} = \begin{cases} [(\alpha_{max,1} - \alpha_{max,2}) + (1 - \sqrt{r_{21}}(1-\alpha_{max,1})\alpha_{max,2}] & \\ [\alpha_{max,1} + (1-\alpha_{max,1})\alpha_{max,2}]\frac{cx_i}{\alpha_{max,1}} + \alpha_{max,2} & \text{if } cx_1 \leq \frac{\alpha_{max,1}}{\alpha_{max,1}+(1-\alpha_{max,1})\alpha_{max,2}} \\ (1-\sqrt{r_{21}})[\alpha_{max,1} + (1-\alpha_{max,1})\alpha_{max,2}]cx_2 + \alpha_{max,1} & \text{else} \end{cases}, \tag{2.89}$$

Where the definitions of r_{21} and cx_i are used from the previous section.

Variable	Description
residualAlpha	Minimum volume fraction to divide by [ ]

2.4.7 Granular drag models

The drag force between granular phases is different than standard drags models due to the fact the the collisions which are a function of the granular energy strongly effect the rate at which the velocities relax. These granular drag models are only necessary when multiple granular phases are used.

2.4.7.1 Chao

The Chao granular drag model (Chao et al., 2011) defines the drag coefficient between phases as

$$K_{D,ij} = \frac{\alpha_i \alpha_j \rho_i \rho_j}{m_0} d_{ij}(1+e_{ij}) g_{0,ij} \tag{2.90}$$

$$\begin{bmatrix} \sqrt{2\pi}(\sqrt{\Theta_i}+\sqrt{\Theta_j}) - \sqrt{2}(\Theta_i\Theta_j)^{1/4} + 0.5\pi|\mathbf{u}_i - \mathbf{u}_j| \\ -1.135\sqrt{|\mathbf{u}_i - \mathbf{u}_j|}\left(\sqrt{4}\Theta_i + \sqrt[4]{\Theta_j} - 0.8\sqrt[8]{\Theta_i\Theta_j}\right) \end{bmatrix} \tag{2.91}$$

2.4.7.2 Syamlal

The Syamlal granular drag model (Benyahia et al., 2012) defines the drag coefficient between phases as

$$K_{D,ij} = \frac{3(1+e_{ij})(\frac{\pi}{2} + \frac{C_f \pi^2}{8})\alpha_i \alpha_j \rho_i \rho_j (d_i + d_j)^2 g_{0,ij} |\mathbf{u}_i - \mathbf{u}_j|}{2\pi(\rho_i d_i^2 + \rho_j d_j^3} + C_1 P_{fr}, \tag{2.92}$$

where C_f is the coefficient of friction between granular phases i and j, and C_1 is the coefficient of frictional pressure.

Variable	Description
C_1	Coefficient of frictional pressure (default is 0) [s/m^2]

3 Turbulence

In addition to the hyperbolic flux terms, viscous effects can be included in simulations. This includes the use of turbulence models such as Reynolds averaged Navier-Stokes (RANS) or Large Eddy simulations (LES) models. All single phase turbulence models standard to OpenFOAM including k-epsilon, k-omega, Smagorinsky, as well as many others are available. The specified model introduces additional transport equations, as well as modifications to transport coefficients like viscosity (ν) and thermal diffusivity (α) which modify the viscous terms in the conservation equations. When the maximum viscosity is zero, the `constant/turbulenceProperties` does not need to be included, however if any phase has viscous terms, this dictionary needs to be included along with the appropriate turbulence model (laminar is the same as no turbulence model). The available turbulence models are listed below, and these can also be found using the "-listTurbulenceModels" option when running `blastFoam`.

Turbulence models: `LES, RAS, laminar` (e.g. no model)
laminar models:

- `Giesekus`
- `Maxwell`
- `Stokes (default)`
- `generalizedNewtonian`

RAS models:

- `LRR`
- `LaunderSharmaKE`
- `RNGkEpsilon`
- `SSG`
- `SpalartAllmaras`
- `buoyantKEpsilon`
- `kEpsilon`
- `kOmega`
- `kOmegaSST`
- `kOmegaSSTLM`
- `kOmegaSSTSAS`
- `realizableKE`
- `v2f`

LES models:

- `DeardorffDiffStress`
- `Smagorinsky`
- `SpalartAllmarasDDES`
- `SpalartAllmarasDES`
- `SpalartAllmarasIDDES`
- `WALE`
- `dynamicKEqn`
- `dynamicLagrangian`
- `kEqn`
- `kOmegaSSTDES`

NOTE: Many of the turbulence models require additional field (`k`, ε, ω, etc.) to be specified in the initial conditions folder.

4 Fluid and solid thermodynamic models

Within *blastFoam* solvers, two different types of basic thermodynamic models are used. The primary is that of a fluid where internal energy and density, or temperature and density, are used to find the resulting pressure and speed of sound. The other, which is only used for solid phases with in multi-fluid solvers and multi-region solvers, is the solid thermodynamic model. These two models will share the basic thermodynamic classes (eConst, hConst), as well as the choice in the type (basic or detonating), however the solid phase only can use a constant density equation of state, and the transport model describes that of heat conduction, rather than flow coefficients (like viscosity). These specific choices will be described below.

To calculate the pressure and temperature given the conservative quantities (mass, momentum, and energy) three components are required: an equation of state, a thermodynamic model, and a transport model. These models are defined in the `phaseProperties` file located in the `constant` directory of each blastFoam case. Each phase within the `phaseProperties` file has several dictionary entries, and these models are specified in the `thermoType` sub-dictionary; the molecular weight must also be defined in the `specie` sub-dictionary.

```
phase_name
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      tabulated;
        equationOfState tabulated;
    }
    specie
    {
        molWeight   28.97;
    }
    transport
    {
        ...
    }
```

```
    thermodynamics
    {
       ...
    }
    equationOfState
    {
        ...
    }
}
```

4.1 Thermodynamic models

The thermodynamic model is used to calculate the internal energy based solely on the thermal contribution:

$$e_t = \int Cv(T)\mathrm{dT}. \tag{4.1}$$

An additional contribution from the equation of state is given by:

$$e_{eos}(\rho_i, T) = \int \left(\frac{\partial p(\rho_i, T)}{\partial T} \right), \mathrm{d}v_i \tag{4.2}$$

where $v_i = 1/\rho_i$ is the specific volume. The total internal energy is therefore

$$e = e_T(T) + e_{eos}(\rho_i, T) \tag{4.3}$$

Because the internal energy is the tracked quantity, the temperature is solved using a Newton-Raphson root finding method given the local internal energy and density.

Either internal energy or enthalpy-based coefficients can be used, but only internal energy is used to define the total energy. The conversion from an enthalpy base to an internal energy base is calculated using the specified equation of state

$$C_v = C_p - CpMCv, \tag{4.4}$$

and

$$e = h - \frac{p_i}{\rho_i}, \tag{4.5}$$

where $CpMCv$ is determined from the equation of state using

$$CpMCv = \frac{\partial(p_i v_i)}{\partial T}. \tag{4.6}$$

For an ideal gas, $CpMCv = \bar{R}/W_i$ where $\bar{R}$ is the ideal gas constant (8314 [J/mol/K]) and W_i is the molecular weight in [kg/kmol]. Equations of state that have a pressure with a non-linear dependence on temperature will deviate from this.

4.1.1 eConst

A constant specific heat at constant volume is specified using the eConst thermodynamic model.

Variable	Description
C_v	Specific at constant volume [J/kg/K]

4.1.2 hConst

A constant specific heat at constant pressure is specified using the hConst thermodynamic model.

Variable	Description
C_p	Specific at constant pressure [J/kg/K]

4.1.3 JANAF

The JANAF thermodynamic model can be used in which the enthalpy is represented by a temperature based polynomial (OpenCFD Ltd., 2018b). Seven coefficients must be specified for both the low temperature and high temperature states, as well as the high, low, and switching temperatures. Currently, this model is only available for ideal gases.

Variable	Description
highCpCoeffs	High enthalpy coefficients
lowCpCoeffs	Low enthalpy coefficients
Tlow	Minimum temperature [K]
Tcommon	Switching temperature [K]
Thigh	Maximum temperature [K]

4.1.4 tabulated

The tabulated thermodynamic model can be used to calculate the temperature by using the density and internal energy to lookup the temperature. The data is read from a two dimensional tables where density is the first component and internal energy is the second.

NOTE: No internal energy or specific heat correction is added from the equation of state, and therefore must be included in the values provided by the table.

Variable	Description
file	Name of file (store in the main case directory)
mod	Modifier of temperature (none, exp, ln, pow10, log10)
rhoMod	Modifier of density(none, exp, ln, pow10, log10)
eMod	Modifier of energy (none, exp, ln, pow10, log10)
nRho	Number of density entries (first direction)
ne	Number of internal energy entries (second direction)
minRho	Minimum density (in given space)
mine	Minimum energy (in given space)
dRho	Density spacing (in given space)
de	Energy spacing (in given space)

4.2 Equation of states

In order to more easily handle a variety of equations of state, both temperature-based equations of state and internal energy based equations of state (e.g. in Mie-Grüneisen form) can be used.

4.2.1 rhoConst

For a solid phase, it is assumed to have a constant density, rho_0, and maintains a reference pressure p_{ref}. Because a solid thermodynamic model is never used to transport mass, the reference pressure is arbitrary and therefore set to 0 [Pa] by default.

Variable	Description
ρ_0	reference density [kg/m^3]
p_{ref}	Reference pressure [Pa] (0 by default)

4.2.2 Internal energy based (e.g. in Mie-Grüneisen form)

The Mie-Grüneisen form uses two contributions to calculate the pressure, and ideal gas term contribution, and a deviation term.

$$p = (\Gamma_i - 1)\rho_i e - \Pi_i, \tag{4.7}$$

Γ_i, also called the Grüneisen coefficient, and Π_i is the pressure deviation from an ideal gas and is specific to the equation of state. Using these quantities, the speed of sound is written as

$$c_i^2 = \frac{\Gamma_i p + \Pi_i}{\rho_i} - \delta_i(\Gamma_i - 1), \tag{4.8}$$

where δ_i is defined

$$\delta_i = -\frac{[p + \Pi_i]\Gamma_i'}{[\Gamma_i - 1]^2} + \frac{\Pi_i'}{\Gamma_i - 1}, \tag{4.9}$$

and where Γ_i' and Π_i' are the derivatives with respect to the phase density.

4.2.2.1 Ideal gas

For an ideal gas, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i. \tag{4.10}$$

The functions as defined to represent an ideal gas in Mie-Grüneisen form are:

$$\Gamma_i = \gamma_i, \tag{4.11}$$

$$\Pi_i = 0, \tag{4.12}$$

$$\delta_i = 0. \tag{4.13}$$

Variable	Description
γ	Specific heat ratio [ ]

4.2.2.2 Stiffened gas

For a material obeying the stiffened EOS, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i a_i, \tag{4.14}$$

where a_i and γ_i are material properties (Zheng et al., 2011). In the Mie-Grüneisen form

$$\Gamma_i = \gamma_i, \tag{4.15}$$

$$\Pi_i = \gamma_i a_i, \tag{4.16}$$

and

$$\delta_i = 0. \tag{4.17}$$

Variable	Description
a	Reference pressure [Pa]
γ	Specific heat ratio [ ]

4.2.2.3 Tillotson

For a material obeying the Tillotson EOS, the pressure is defined as

$$p_i = p_{0,i} + \omega_i \rho_i (e_T - e_{0,i}) + A_i \mu + B_i \mu^2 + C_i \mu^3, \tag{4.18}$$

where A_i, B_i, C_i, $omega_i$, and γ_i are material coefficients, and $p_{0,i}$ and $e_{0,i}$ are the reference pressure and internal energy (Wardlaw et al., 1998). μ is defined as $\rho_i / rho_{0,i} - 1$.

A simplified cavitation model is also included where $p_i = p_{cav}$ when $\rho_i < rho_{0,i}$. The total energy is defined as a combination of the thermal energy , $e_{thermal} + e_{cav}$, where the cavitation energy is found by integrating

$$\frac{\partial e_{cav}}{\rho_i} = \frac{p_i(\rho, e_{cav})}{\rho^2}, \tag{4.19}$$

with the initial conditions $\rho = \rho_0$ and $e_{cav} = 0$.

In the Mie-Grüneisen form

$$\Gamma_i = \omega_i + 1, \tag{4.20}$$

and

$$\Pi_i = \omega_i \rho_i e_{0,i} - p_{0,i} - A_i \mu - B_i \mu^2 - C_i \mu^3) \tag{4.21}$$

and

$$\delta_i = e_{0,i} - \frac{A + 2B\mu + 3.0C\mu^2}{\rho_{0,i}\omega_i}. \tag{4.22}$$

Variable	Description
p_0	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
e_0	Reference internal energy [kJ/kg]
A	Model coefficient [Pa]
B	Model coefficient [Pa]
γ	Model coefficient [ ]
ω	Model coefficient [ ]
pCav	Cavitation pressure [Pa]
rhoCav	Cavitation density [kg/m^3]

4.2.2.4 Tait

For a material obeying the stiffened EOS, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i (b_i - a_i Z z, \tag{4.23}$$

where a_i, b_i, and γ_i are material properties (Zheng et al., 2011). In the Mie-Grüneisen form

$$\Gamma_i = \gamma_i, \tag{4.24}$$

$$\Pi_i = \gamma_i (b_i - a_i), \tag{4.25}$$

and

$$\delta_i = 0. \tag{4.26}$$

Variable	Description
a	Material parameter [Pa]
b	Material parameter [Pa]
γ	Specific heat ratio [ ]

4.2.2.5 Van der Waals

For a gas described by the generalized van der Waals equation of state (Zheng et al., 2011), the pressure is defined as

$$p_i = \frac{\gamma_i - 1}{1 - b_i \rho_i}(\rho_i e_i + a_i \rho_i^2) - (a_i \rho_i^2 + c_i). \tag{4.27}$$

Putting the pressure in the Mie-Grüneisen form, we obtain

$$\Gamma_i = \frac{\gamma_i - 1}{1 - b_i \rho_i} + 1, \tag{4.28}$$

$$\Pi_i = \left[1 - \frac{\gamma_i - 1}{1 - b_i \rho_i}\right] a_i \rho_i^2 + \left[\frac{\gamma_i - 1}{1 - b_i \rho_i} + 1\right] c_i, \tag{4.29}$$

and

$$\delta_i = -b_i \frac{p_i + a_i \rho_i^2}{\gamma_i - 1} + \left(\frac{1 - b_i \rho_i}{\gamma - 1} - 1\right) 2 a_i \rho_i. \tag{4.30}$$

Variable	Description
a	Attraction between particles [$\mathrm{kg^{-1}\ m^5\ s^{-2}}$]
b	Specific volume excluded due to particle volume [$\mathrm{kg^{-1}\ m^3}$]
c	Reference pressure [Pa]
γ	Specific heat ratio [ ]

4.2.2.6 Landau, Stanyukovich, Zeldovich, and Kompaneets (LSZK)

The LSZK equation of state is a simple model for describing an ideal explosive (Lutzky, 1964; Needham, 2018). It takes a form similar to a stiffened gas or Tait EOS, with

$$\Gamma_i = \gamma_i, \tag{4.31}$$

$$\Pi_i = a_i \rho_i^{b_i}, \tag{4.32}$$

and

$$\delta_i = 0. \tag{4.33}$$

Variable	Description
a	Model constant [$\mathrm{kg}^{1-b}\mathrm{s}^{-2}\mathrm{m}^{3b-1}$]
b	Pressure exponent []
γ	Specific heat ratio [ ]

4.2.2.7 Jones Wilkins Lee (JWL)

The more complicated JWL EOS is often used to define energetic materials (Zheng et al., 2011), and has a pressure deviation given by

$$\Pi_i = A_i e^{-R_{1,i}V}\left(1.0 - \frac{\omega_i}{R_{1,i}V}\right) + B_i e^{-R_{2,i}V}\left(1.0 - \frac{\omega_i}{R_{2,i}V}\right) \tag{4.34}$$

where $V = \rho_{0,i}/\rho_i$.

$$\Gamma_{r,i} = \omega_i + 1, \tag{4.35}$$

and

$$\delta_{r,i} = \tag{4.36}$$

$$A_i e^{-\frac{R_{1,i}\rho_{0,i}}{\rho_i}}\left[\omega_i\left(\frac{1}{R_{1,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{1,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i} \tag{4.37}$$

$$+B_i e^{-\frac{R_{2,i}\rho_{0,i}}{\rho_i}}\left[\omega_i\left(\frac{1}{R_{2,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{2,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i}. \tag{4.38}$$

NOTE: The JWL model can also be used to describe the unreacted state if a detonating model is used with the "solidJWL" type.

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
R_1	Model coefficient [ ]
R_2	Model coefficient [ ]
ρ_0	Unreacted or reference density [kg m^{-3}]
ω	Grüneisen coefficient [ ]

4.2.2.8 Cochran-Chan

The Cochran Chan EOS can be used to describe a solid material (Zheng et al., 2011), and has a reference pressure given by

$$p_{ref,i} = A_i V_i^{1-\mathcal{E}_{1,i}} - B_i V_i^{1-\mathcal{E}_{2,i}}. \tag{4.39}$$

The functions used to defined the Mie-Grüneisen equations are

$$\Gamma_i = \Gamma_{0,i} + 1, \tag{4.40}$$

$$\Pi_i = \Gamma_{0,i}\rho_i \left(-\frac{A_i}{(\mathcal{E}_{1,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{1,i}} + \frac{B_i}{(\mathcal{E}_{2,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{2,i}} \right) - p_{ref,i}, \tag{4.41}$$

and

$$\delta_i = \tag{4.42}$$

$$\frac{A_i}{\mathcal{E}_{1,i}} \left[\mathcal{E}_{1,i} V_i^{-\mathcal{E}_{1,i}} \frac{\mathcal{E}_{1,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}} \tag{4.43}$$

$$+ \frac{B_i}{\mathcal{E}_{2,i}} \left[\mathcal{E}_{2,i} V_i^{-\mathcal{E}_{2,i}} \frac{\mathcal{E}_{2,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}}. \tag{4.44}$$

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
$\mathcal{E}_1$	Model coefficient [ ]
$\mathcal{E}_2$	Model coefficient [ ]
ρ_0	Reference density [kg/m^3]
Γ_0	Grüneisen coefficient [ ]

4.2.2.9 Doan-Nickel

The Doan-Nickel EOS includes changes in the composition of air at high temperatures and the changing specific heat ratio due to dissociation of molecules Doan and Nickel (1963). Due to the way this model is defined, it can currently only be used with inviscid fluids and a constant specific heat at constant volume (eConst). The specific heat is a function of the density and internal energy and is defined as

$$\Gamma_i = 1+\left[0.161 + 0.255e^{\tilde{e}/e_1}f_1 + 0.280e^{\tilde{e}/e_2}(1-f_1) + 0.137e^{\tilde{e}/e_3}f_2 + 0.050f_3\right]\left(\frac{\rho_i}{\rho_{0,i}}\right)^{\alpha(\rho_i,\tilde{e})} \tag{4.45}$$

where $\tilde{e} = e \cdot 10^{-6}$. $e_1 = 4.46$, $e_2 = 6.63$, and $e_3 = 25.5$ are the internal energies at which oxygen dissociates, everything is dissociated, and electrons contribute, respectively. $\rho_{0,i}$ is the density of air at sea-level (1.293 [kg/m^3], and

$$\alpha(\rho_i, \tilde{e}) = [0.048f_1 + 0.032(1-f_1)(1-f_2)]\log_{10}(\tilde{e}) + 0.045f_3, \tag{4.46}$$

with

$$f_1 = \frac{1}{\exp\left(\frac{\tilde{e}-e_{O_2->2O}}{\Delta e_{O_2->2O}}\right)+1}, \tag{4.47}$$

$$e_{O_2->2O} = 8.5 + 0.357\log_{10}\left(\frac{\rho_i}{\rho_{0,i}},\right) \tag{4.48}$$

$$\Delta e_{O_2->2O} = 0.975\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.05}, \tag{4.49}$$

$$f_2 = \frac{1}{\exp(\frac{e_{N_2->2N}-\tilde{e}}{\Delta e_{N_2->2N}})+1}, \tag{4.50}$$

$$e_{N_2->2N} = 45.0\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.0157}, \tag{4.51}$$

$$\Delta e_{N2->2N} = 4.0\left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.085}, \tag{4.52}$$

$$f_3 = \frac{1}{\exp(\frac{e_e-\tilde{e}}{\Delta e_e})+1}, \tag{4.53}$$

$$e_e = 160.0, \tag{4.54}$$

and

$$\Delta e_e = 6.0. \tag{4.55}$$

Due to the complexity of the model, and the fact that it was *specifically* developed as a model for air, all coefficients are hard-coded and cannot be changed. This equation of state should only be used to represent standard air.

4.2.3 Temperature-based

While it is preferable to use an internal energy based equation of state for compressible flows, some equations of state cannot be converted, and therefore use the local temperature to define the thermodynamic pressure. Generally the Grüneisen coefficient used to calculate the shared pressure is defined as

$$\Gamma_i = \frac{v_i}{C_v}\frac{\partial p}{\partial T}, \tag{4.56}$$

where C_v is the specific heat at constant volume, and the speed of sound is defined

$$c_i^2 = v_i^2 \left[\left(\frac{\partial p_i}{\partial T} \right)^2 \frac{T}{C_v} - \frac{\partial p}{\partial v_i} \right]. \tag{4.57}$$

Unless otherwise specified, these relations are used to calculate the relevant quantities.

4.2.3.1 Jones Wilkins Lee C-form (JWLC)

The C-form of the JWL equation of state (Souers et al., 2000) does not have a dependence on the temperature or internal energy and has a pressure defined as

$$p_i = A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i}{V^{1+\omega_i}}, \tag{4.58}$$

the speed of sound if defined

$$c_i^2 = \frac{V^2}{\rho_{0,i}} \left(A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i(1+\omega_i)}{V^{2+\omega_i}} \right). \tag{4.59}$$

The Grüneisen coefficient is the same as the standard JWL equation of state.

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
C	Model coefficient [Pa]
R_1	Model coefficient [ ]
R_2	Model coefficient [ ]
ρ_0	Unreacted or reference density [kg m^{-3}]
ω	Grüneisen coefficient [ ]

4.2.3.2 Abel-Nobel (AbelNobel)

The equation of state of Abel-Nobel is commonly used for modeling simulate explosive materials, specifically for propellants (Johnston, 2005). Unlike the JWL equation of state, the BKW models uses the temperature, rather than the internal energy, to define the pressure.

$$p_i = \frac{RT\rho_i}{1 - b_i\rho_i} \tag{4.60}$$

where b is the co-volume of the fluid.

Variable	Description
b	Co-volume [m^3/kg]

4.2.3.3 Becker-Kistiakowsky-Wilson (BKW)

The BKW equation of state is another equation of state that is commonly used to simulate explosive materials (Mader, 1963). Unlike the JWL equation of state, the BKW models uses the temperature, rather than the internal energy, to define the pressure.

$$p_i = RT\rho_i \left(1 + xe^{\beta x}\right) \tag{4.61}$$

where

$$x = \frac{\kappa k \rho_i}{W_i (T + \theta)^a} \tag{4.62}$$

and W_i is the molecular weight in [kg/kmol].

Variable	Description
β	Model coefficient [ ]
a	Temperature exponent [ ]
Θ	Model coefficient [K]
κ	Model coefficient [K^a]
k	Specie co-volume [m^3/kmol]

4.2.3.4 Benedict-Webb-Rubin (BWR)

The BWR equation of state was developed by (Benedict et al., 1942) to describe real gases using a set of eight coefficients that must be experimentally determined. The pressure is defined as

$$\begin{aligned} p_i =& RT\rho_i + \left(B_0RT - A_0 - \frac{C_0}{T^2}\right)\rho_i^2 + (bRT - a)\rho_i^3 + a\alpha\rho_i^6 \\ &+ c\frac{\rho_i^3}{T^2}(1 + \gamma\rho_i^2)e^{-\gamma\rho_i^2} \end{aligned} \tag{4.63}$$

Variable	Description
A_0	Model coefficient
B_0	Temperature exponent
C_0	Model coefficient
a	Model coefficient
b	Model coefficient
c	Model coefficient
α	Model coefficient
γ	Specific heat ratio [ ]

4.2.3.5 Murnaghan

The Murnaghan (or Munahan) equation of state is used for solid material and has no temperature dependence (Souers et al., 2000). The pressure is a function of only the density and is defined as

$$p_i = p_{ref} + \frac{1}{\kappa n}\left(\frac{\rho_i}{\rho_{0,i}} - 1\right)^n. \tag{4.64}$$

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
κ	Model coefficient [Pa^{-1}]
n	Model coefficient [ ]
Γ	Grüneisen coefficient [ ]

4.2.3.6 Birch-Murnaghan (2nd order)

The second order Birch-Murnaghan equation of state is the second order expansion of the Birch-Murnaghan model. The pressure is defined as

$$p_i = p_{ref} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right] \tag{4.65}$$

Many coefficients can be found for this in Zheng and Zhao (2016).

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
K_0	Bulk modulus [Pa]
Γ	Grüneisen coefficient [ ]

4.2.3.7 Birch-Murnaghan (3rd order)

The Third order Birch-Murnaghan equation of state is the third order expansion of the Birch-Murnaghan (see Zheng and Zhao (2016)). The pressure is defined as

$$p_i = p_{ref} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right]\left\{1 + 0.75(K_0' - 4)\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{2/3} - 1\right]\right\} \tag{4.66}$$

Variable	Description
p_{ref}	Reference pressure [Pa]
ρ_0	Reference density [kg/m^3]
K_0	Bulk modulus [Pa]
K_0'	Change in bulk modulus w.r.t. pressure [ ]
Γ	Grüneisen coefficient [ ]

4.3 Tabulated equation of state (tabulated)

In addition to the previously mentioned equations of state and thermodynamic models, tabulated models may also be used. This is useful when simulating real gasses, and high temperatures and pressures when standard models are no longer sufficient. Any combination of pressure and temperature tables can be used, as long as they are a function of density and internal energy (x and y, respectively). The internal energy is initialized using a reverse look up given a pressure and a density. Modifiers can also be used such as: exponential, natural log, and log base 10. The minimum values of density and internal energy, as well as the spacing, both in the modified values (i.e. $\log(\rho_{min})$ for a density in natural log units). The number of rows and columns must also be specified (nRho and ne). All necessary derivatives are calculated using the finite difference method. An example is shown below:

```
phase
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      tabulated;
        equationOfState tabulated;
    }
    specie
    {
        molWeight   28.97;
    }
    thermodynamics
    {
        file    "T.csv";
```

```
        mod     ln;     // log(p_i)=p_table

        nRho    7;      // Number of columns
        rhoMod  log10;  // log_10(rho_i)=minRho+i*dRho
        minRho  -3.0;   // 0.001 kg/m^3
        dRho    1.0;

        ne      40;     // Number of rows
        eMod    ln;     // log(e)=mine+j*de
        mine    11.8748;
        dRho    1.0;
    }
    equationOfState
    {
        file    "p.csv";
        mod     ln;     // log(p_i)=p_table

        nRho    7;      // Number of columns
        rhoMod  log10;  // log_10(rho_i)=minRho+i*dRho
        minRho  -3.0;   // 0.001 kg/m^3
        dRho    1.0;

        ne      40;     // Number of rows
        eMod    ln;     // log(e)=mine+j*de
        mine    11.8748;
        dRho    1.0;
    }
}
```

Variable	Description
file	Name of file
mod	Modifier of the data (log10, ln, exp)
nRho	Number of columns
ne	number of rows
rhoMod	Density modifier
eMod	internal energy modifier
dRho	Density spacing in modified units
de	internal energy spacing in modified units

4.4 Fluid Transport

The transport models are used to define quantities such as viscosity and thermal diffusion. Only Newtonian viscosity models are currently available.

4.4.1 Constant transport (const)

Constant values for viscosity (μ) and Prandtl number are used. These values are used to calculate the thermal diffusion coefficient using $\alpha_T = \mu/Pr$.

Variable	Description
μ	Dynamic viscosity [Kg/m/s]
Pr	Prandtl number [ ]

4.5 Solid Transport

The transport models are used to define the thermal conductivity of the material

4.5.1 Constant isotropic transport (constIso)

A constant, scalar value for thermal conductivity is used (κ). These values are used to calculate the thermal diffusivity coefficient using $\alpha_T = \kappa/C_p$ where C_p is determined using the thermodynamic model.

Variable	Description
κ	Thermal conductivity [W/m/K]

4.5.2 Constant anisotropic transport (constAnIso)

A constant, vector value for thermal conductivity is used ($\boldsymbol{\kappa}$). These values are used to calculate the thermal diffusivity coefficient using $\boldsymbol{\alpha_T} = \boldsymbol{\kappa}/C_p$ where C_p is determined using the thermodynamic model.

Variable	Description
$\boldsymbol{\kappa}$	Thermal conductivity (vector) [W/m/K]

4.6 Basic thermodynamic model

The `basicThermoFluid` model and `basicSolid` model (type `basic`) uses a transport model, thermodynamic model, and equation of state to describe a phase. The basic model should be used for non-reacting phases.

4.7 Detonating thermodynamic model

In order to better describe a detonating material, a detonating thermodynamic model can be used. This model transitions between an unreacted state and a reacted state using an activation model. Each state (unreacted and reacted) is described by its own basic thermodynamic model, allowing for different models and coefficients to be used for values such as viscosity or specific heats. In addition to the activation model, afterburn models can also be used to add additional energy into the system after the initial detonation reaction has taken place.

This same methodology is applied to solid phases where energy is released locally to increase the thermal energy (internal energy). Only activation models are currenltly available.

Some standard combinations of unreacted and reacted equations of states include the `JWL++` (`Murnaghn` and `JWLC`), and the `blendedJWL` (`solidJWL` and `JWL`).

NOTE: Whilst it will not give the correct temperature, using an ideal gas is a simple option if the solid properties of the unreacted material are not known.

```
phase1
{
    type            detonating;
```

```
    reactants
    {
        ...
    }
    products
    {
        ...
    }

    activationModel linear;
    initiation
    {
        E0      9.0e9;
        points  ((0 0 0) (0.1 0.1 0.1));
        delays  (0 1e-5);
        radii   (0.001 0.005);
        vDet    5000;
    }

    afterburnModel Miller;
    MillerCoeffs
    {
        Q0      1e7;
        m       0.1667;
        n       0.5;
        a       0.0195e6;
        pMin    1.1e5;
    }
}
```

4.7.1 Activation Models

In order to accurately model the release of energy, different models can be used to approximate the speed at which the solid reactants become the product gases. The currently available models include the linear model, a pressure based models, and finally an Arrhenius rate reaction. Theses models all use a reaction progress variable, λ_i, to describe the state of individual cell activation. Additionally, in order to add numerical stability, the initial energy is

defined using the unreacted equation of state, and the reaction energy, E_0, is added based on the time rate of change of the reaction progress variable, i.e.

$$\dot{E} = \frac{E_0 \rho_i}{\rho_{0,i}} \frac{d\lambda}{dt} \tag{4.67}$$

The phase properties are calculated using

$$x_i = \lambda_i^l x_{r,i} + (1 - \lambda_i^l) x_{u,i)} \tag{4.68}$$

where $x_{r,i}$ is the reacted phase quantity, $x_{u,i}$ is the unreacted phase quantity, and l is the exponent used to blend the unreacted and reacted states (lambdaPow). By default l is one. The advection scheme and limiters for λ_i need to be specified in the `fvSchemes` in the same manner as the other transported quantities.

The setting of detonation points can be done for any activation model. By setting detonation points the area within a charge, the cells with the radii specified is set to full activated when run time reached the given delays. If no radii are provided only the cell containing the point will be activated. Additionally, delays can be used to specify the time at which the given point and cells within the radius are activated. For a single detonation point, the *useCOM* option can be used to calculate the center of mass of an individual charge.
The reaction progress variable is transported using

$$\frac{\partial \alpha_i \rho_i \lambda_i}{\partial t} + \boldsymbol{\nabla} \cdot (\alpha_i \rho_i \mathbf{u} \lambda_i) = \alpha_i \rho_i \dot{R}_i \tag{4.69}$$

where $\dot{R}_i$ is the reaction rate determined by the model of choice.

Variable	Description
E_0	Detonation energy [Pa]
points	Detonation points [m]
radii (radius)	Initiation radii around detonation points (default is 0) [m]
delays	Delay time of detonation points (default is 0) [s]
lambdaPow	Exponent used for blending [ ] (default is 1)
useCOM	Is the center of mass used as the detation point (default is no)

4.7.1.1 No Activation (none)

The `None` activation model assumes that all cells are activated when the

simulation begins. This is the equivalent to an infinitely fast detonation velocity. The use of delays and detonation points are not valid here because the detonation is assumed to be instantaneous. There is no transport of the reaction progress variable.

4.7.1.2 Linear Activation (linear)

The most simple activation model uses a list of detonation points and a constant detonation velocity to determine which cells have been activated. This gives

$$\lambda_{i,j} = \begin{cases} 1 & \text{if } |\mathbf{x} - \mathbf{x}_{\mathrm{det,i,j}}| < \mathrm{v}_{\mathrm{det,i}} t \\ 0 & \text{else} \end{cases} \tag{4.70}$$

where $\mathbf{x}$ is the cell center, $\mathbf{x}_{\mathrm{det,i,j}}$ is the j^{th} detonation point, $\mathrm{v}_{\mathrm{det,i}}$ is the detonation velocity, and t is the time. In order to account for multi-point detonation the actual λ is defined as

$$\lambda_i = \max(\lambda_{i,j}). \tag{4.71}$$

The reaction progress variable can be transported if the *advection* option is set to "on". This is recommend due to the fact that small amounts of mass can move out of a cell, causing additional (nonphysical) energy to be added the domain.

Variable	Description
vDet	Speed of propagation within the material [m/s]
advection	Is advection of the reaction progress variable used (default yes)

4.7.1.3 Programmed ignition (programmedIgnition)

The *programmedIgnition* activation model is slightly more complex than the linear activation model and included the option to include activation due to compression of material, and is based on the work of Giroux (1973). The reaction progress variable is defined using a combination of two models, the compression

$$\lambda_{\beta,i} = \frac{1 - \frac{1}{\rho_i}}{1 - V_{CJ,i}} \tag{4.72}$$

where $V_{CJ,i}$ is the Chapman-Jougnuet specific volume which is compututed using the Chapman-Jouguet pressure. The programmed definition of the

reaction progress variable is a modified version of the linear activation where a cell reacts a time-rate determined by the detonation velocity and the local grid size

$$\lambda_{programmed,i} = \max(t - t_{ign,j}, 0)\frac{v_{det}}{1.5dx} \tag{4.73}$$

where $\mathbf{t_{ign}}$ is the time of ignition determined by the physical location and the detonation velocity, and dx is the local grid size.
if the *beta* burn option is used, $\lambda_i = \lambda_{\beta,i}$; if the *programmed* burn option is used, $\lambda_i = \lambda_{programmed,i}$; and if the *programmedBeta* burn option is used $\lambda_i = \max(\lambda_{i,\beta}, \lambda_{programmed,i}$.

Variable	Description
vDet	Speed of propagation within the material [m/s]
Pcj	Chapman-Jouguet pressure [Pa]
burnModel	Burn model used (beta, programmed, or programmedBeta)

4.7.1.4 Pressure based activation (pressureBased)

A more advanced activation model uses the local cell pressure, determined by the equation of state, and follows the evolution equation

$$\dot{R}_i = \mathrm{I}\left(\frac{\rho_i}{\rho_{0,i}} - 1 - a\right)^x (1-\lambda)^b + \mathrm{G}_1(1-\lambda)^c\lambda^d p^y + \mathrm{G}_2(1-\lambda)^e\lambda^f p^z \tag{4.74}$$

The first term is responsible for the initial detonation based on the compression of the material. The following two terms account for the subsequent reaction based on the local pressure. All terms can be turned "on" or "off" by specifying the minimum and maximum values of lambda at which the stages occur.

Variable	Description
I	Ignition rate [s^{-1}]
a	Compression ratio offset [ ]
b	Ignition $(1-\lambda)$ exponent [ ]
x	Compression ratio exponent [ ]
maxLambdaI	Maximum value of λ that the ignition term is used [ ]
G_1	First activation rate [$Pa^{-y}s^{-1}$]
c	First $(1-\lambda)$ exponent [ ]
d	First λ exponent [ ]
y	First pressure exponent [ ]
minLambda1	Minimum value of λ that the first step is used [ ]
maxLambda1	Maximum value of λ that the first step is used [ ]
G_2	Second activation rate [$Pa^{-z}s^{-1}$]
e	Second $(1-\lambda)$ exponent [ ]
f	Second λ exponent [ ]
z	Second pressure exponent [ ]
minLambda2	Minimum value of λ that the second step is used [ ]
maxLambda2	Maximum value of λ that the second step is used [ ]

4.7.1.5 Arrhenius rate activation (ArrheniusRate)

A more physical representation of the activation rate uses the local temperature, defined by the internal energy and the specific heat of the mixture. The evolution equation for the reaction progress variable is

$$\dot{R}_i = (1-\lambda)\begin{cases} 0 & T < T_{ign} \\ \mathrm{d_p^2 A_{low} exp}(\frac{E_{\mathrm{a,low}}}{RT}) & \text{if } T < T_s \\ \mathrm{A_{high} exp}(\frac{E_{\mathrm{a,high}}}{RT}) & \text{else} \end{cases}. \tag{4.75}$$

Where the particle diameter is determined by the chosen diameter model (see Sec. 5), and R is the ideal gas constant.

Variable	Description
diameterModel	Diameter model used to determine the particle diameter
A_{low}	Low activation rate [m^{-2} s^{-1}]
$E_{a,low}$	Low activation energy [m^2/s^2]
A_{high}	High activation rate [s^{-1}]
$E_{a,high}$	High activation energy [m^2/s^2]
T_s	Switch temperature [K]
T_{ign}	Minimum temperature of reaction

4.7.2 Afterburn Models

When reactions occur over a period of time after the initial charge has been activated, additional energy can be added that results in higher pressure and temperature. This is referred to as *afterburn*. Afterburn is typical in under-oxygenated and/or non-ideal explosives, which may release energy at a later time. The advection scheme and limiters for c_i need to be specified in the `fvSchemes` in the same manner as the other transported quantities.

4.7.2.1 No Afterburn (none [default])

No additional energy is added. If the keyword *afterburnModel* is not found in the phase dictionary, this is the model that is used.

4.7.2.2 Constant Afterburn (constant)

A constant amount of afterburn energy is added where $\dot{Q} = Q_0$.

Variable	Description
Q_0	Afterburn energy [J/kg/s]

4.7.2.3 Linear Afterburn (linear)

Afterburn energy is added linearly from a start time to an end time where

$$\dot{Q} = \begin{cases} \frac{Q_0}{\Delta t} & \text{if } t_{start} < t < t_{end} \\ 0 & \text{else} \end{cases}. \tag{4.76}$$

Variable	Description
Q_0	Afterburn energy [J/kg]
t_{start}	Time energy begins being added [s]
t_{end}	Time energy stops being added [s]

4.7.2.4 Miller Extension Afterburn Model (Miller)

The model of Miller (1995) uses an evolution equation to determine the fraction of the total amount of afterburn energy added to the system due to unburnt material. c_i is the reaction progress variable that has a value between 0 and 1, with an evolution equation defined as

$$\frac{\partial \alpha_i \rho_i c_i}{\partial t} + \boldsymbol{\nabla} \cdot (\alpha_i \rho_i \mathbf{u} c_i) = \alpha_i \rho_i a (1 - c_i)^m p^n, \tag{4.77}$$

and the total afterburn energy is found using

$$\dot{Q} = \frac{\mathrm{d}c_i}{\mathrm{d}t} Q_0. \tag{4.78}$$

Variable	Description
a	Model constant [Pa^{-n}]
m	Reaction progress variable exponent [ ]
n	Pressure exponent [ ]
Q_0	Afterburn energy [J/kg]
p_{min}	Minimum pressure that afterburn occurs [Pa]

4.8 Two/multiphase models

To simulate a two-phase or multiphase system, the keyword *phases* must be defined in the `phaseProperties` dictionary and defines the list of phases used in a simulation. The selection of a two-phase or multiphase system is determined by the number of defined phases. The mixture pressure is defined by

$$p = \frac{\sum_i \alpha_i \xi_i p_i}{\sum_i \alpha_i \xi_i}, \tag{4.79}$$

where

$$\xi_i(\rho_i) = \frac{1}{\Gamma_i - 1}, \tag{4.80}$$

and p_i is the pressure from a single equation of state (Zheng et al., 2011). The speed of sound within a given phase is given by

$$c_i = \sqrt{\frac{\sum_i y_i \xi_i c_i^2}{\sum_i \xi_i}} \tag{4.81}$$

with

$$y_i = \frac{\alpha_i \rho_i}{\rho}. \tag{4.82}$$

4.9 Initialization

The initialization of fields can be handled in several ways, but ρ_i, p, and T must be given as initial conditions. Some of these fields may be overwritten, but the boundary conditions are used to construct the conserved quantities. For equations of state in the Mie-Grüneisen form, by default the pressure and density are used to initialize the internal energy. Temperature based equations of state initialize the internal energy using the density and temperature. If the *calculateDensity* keyword is defined, and set to *yes* in the given equation of state dictionary the density is calculated from the pressure and temperature prior to the initialization of the internal energy. Each of the initialization methods use a Newton-Raphson root finding method to find the necessary quantities. If the internal energy field is provided in the initial conditions folder, the pressure and temperature fields are calculated directly from the density and internal energy.

4.10 Example

An example of a single phase phaseProperties can be seen below

```
mixture
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
```

```
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cv      718;
        Hf      0.0;
    }
    idealGas
    {
        gamma   1.4;
    }
}
```

For a two or multiphase simulation, the dictionary is defined as

```
phases (air water);
air
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cv      718;
        Hf      0.0;
```

```
    }
    equationOfState
    {
        gamma   1.4;
    }
}
water
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      hConst;
        equationOfState stiffenedGas;
    }
    specie
    {
        molWeight   18.0;
    }
    transport
    {
        mu      0.0;
        Pr      1.0;
    }
    thermodynamic
    {
        Cp      4186;
        Hf      0.0;
    }
    equationOfState
    {
        gamma   4.4;
        a       7e8;
    }
}
```

5 Diameter models

The diameter of a dispersed phase is used to calculate surface reaction rates in the Arrhenius activation model as well as interfacial transfer sources in the *blastEulerFoam* solver. Below are is a short description of the models currently available.

5.1 Constant diameter (constant)

A constant diameter is used.

Entry	Description
d	Diameter of the phase [m]

5.2 Constant mass (constantMass)

The diameter is determined using a reference diameter at a reference density. The mass is defined as

$$M = \frac{\pi d_0^3 \rho_0}{6} \tag{5.1}$$

and the true diameter is calculated using

$$d = \left(\frac{6M}{\rho\pi}\right)^{1/3} \tag{5.2}$$

Entry	Description
ρ_0	Reference density [kg/m^3]
d_0	Reference diameter [m]

5.3 Quadrature based diameter (qbmm)

If the OpenQBMM library has been included, the qbmm diameter model can be used. Here the mean diameter of the size distribution is used. This diameter is much more involved than the previous models. The initial size moments must be provided as inputs to the simulation, these can be initialized by hand, or using the *generateMoments* utility from OpenQBMM. This

model can include size changes due to aggregation, breakup, growth/evaporation, and nucleation as well as moment diffusion. It is highly recommend to familiarize yourself with the method before using this model. An overview of the univarite moment transport section of the library can be seen in Passalacqua et al. (2018).

6 Flux Evaluation

blastFoam relies on explicit solutions for the evolution of conservative variables. This has several benefits in terms of order of accuracy and computational cost. First, higher-order, explicit Runge-Kutta time integration schemes can be used which allow for fully third order solutions to be obtained. Secondly, higher order interpolation schemes, such as cubic, are marginally more computationally expensive than the standard linear interpolation. This is very different than the higher-order divergence, gradient, and Laplacian schemes which are used in implicit solution methods and are significantly more expensive than their linear counterparts. This allows for use of higher order schemes without the downside of significant additional computational cost.

All of the methods currently available to calculate the conservative, hyperbolic fluxes rely on the owner (own) and neighbour (nei) interpolated values on a face (also called left and right states). These interpolated values are found using a combination of a base interpolation scheme (linear or cubic) and flux limiters. OpenFOAM currently has a wide selection of available limiters such as upwind, Minmod, vanLeer, SuperBee, etc. for scalars, and upwind, MinmodV, vanLeerV, SuperBeeV, etc. for vectors; all are run-time selectable.

In addition to the standard OpenFOAM interpolation schemes, MUSCL reconstruction has also been implemented. There are three main options which include `upwindMUSLC`, `linearMUSCL`, and `quadraticMUSCL`. The upwind version uses standard upwinding for owner and neighbour interpolations. Both the linear and quadratic version use limiters in combination with the gradients (linear and quadratic) and the gradient of the gradients (quadratic) to interpolate values. All of the standard limiters are available for MUSCL reconstruction.
An example of the `fvSchemes` dictionary is shown below

```
fluxScheme HLLC;

ddtSchemes
{
    default Euler;
    timeIntegrator RK2SSP;
}
```

```
gradSchemes
{
    default cellMDLimited leastSquares 1.0;
}

divSchemes
{
    default none;
}

laplacianSchemes
{
    default Gauss linear corrected;
}

interpolationSchemes
{
    default cubic; //Base interpolation scheme

    // Limiters
    reconstruct(alpha)  vanLeer;                // Volume fraction
    reconstruct(rho)    upwindMUSCL;            // Density
    reconstruct(p)      linearMUSCL    vanLeer; // Pressure
    reconstruct(U)      quadraticMUSCL vanLeer; // Velocity
    reconstruct(e)      vanLeer;                // Internal energy
    reconstruct(speedOfSound) vanLeer;          // Speed of sound
}
```

6.1 Riemann Solvers

The currently available Riemann solvers are described below. For all schemes, $\mathbf{n}$ denotes the surface normal vector, and $u = \mathbf{u} \cdot \mathbf{n}$. There is the inclusion of volume fraction in the solution of multi-fluid problems, however with the exception of the AUSM+up flux scheme for granular phases these changes will not be included in the descriptions below.

6.1.1 HLL

The HLL scheme is based on Toro et al. (1994) in which the contact wave is neglected. The fluxes take the form:

$$\mathbf{F}^{HLL} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \frac{S_{nei}\mathbf{F}_{own} - S_{own}\mathbf{F}_{nei} + S_{own}S_{nei}(\mathbf{U}_{nei} - \mathbf{U}_{own})}{S_{nei} - S_{own}} & \text{if } S_{own} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{6.1}$$

The owner and neighbor states to approximate the wave propagation speeds are defined as

$$S_{own} = \min(u_{own} - c_{own}, \tilde{u} - \tilde{c}), \tag{6.2}$$

$$S_{nei} = \max(u_{nei} + c_{nei}, \tilde{u} + \tilde{c}), \tag{6.3}$$

where u_K is the normal flux of the respective state, c_K is the speed of sound, and $\tilde{u}$ and $\tilde{c}$ are the Roe averages velocities and speed of sounds, respectively.

6.1.2 HLLC

The approximate HLLC Riemann was developed by Toro et al. (1994), and improves upon the HLL method by providing an estimation of the contact wave between the owner and neighbour waves. This means that the state between the owner and neighbour waves now has two states rather than one. The following fluxes are thus used, using the fact that both pressure and velocity are constant across the contact wave,

$$\mathbf{F}^{HLLC} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \mathbf{F}_{*own} & \text{if } S_{own} \leq 0 \leq S_{*} \\ \mathbf{F}_{*nei} & \text{if } S_{*} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{6.4}$$

The contact wave speed is given by

$$S_{*} = \frac{\rho_{own}u_{own}(S_{own} - U_{own}) + \rho_{nei}u_{nei}(S_{nei} - U_{nei}) + p_{nei} - p_{own}}{\rho_{own}(S_{own} - u_{own}) - \rho_{nei}(S_{nei} - u_{nei})} \tag{6.5}$$

and the pressure in the

$$p_{*K} = \rho_K(S_K - U_K)(S_* - u_K) + p_K \tag{6.6}$$

The final fluxes are determined by

$$\mathbf{F}_{*K}^{HLLC} = \frac{S_*(S_K\mathbf{U}_K - \mathbf{F}_K) + S_K p_{*K}\mathbf{D}_*}{S_K - S_*} \tag{6.7}$$

$$\mathbf{D}_* = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{n} \\ S_* \end{pmatrix}. \tag{6.8}$$

6.1.3 HLLC-P (HLLCP)

Following the work of Xie et al. (2019), a low-speed correction of the standard HLLC flux scheme has been added. Currently it is only available for single phase simulations.

$$\mathbf{F}^{HLLC-P} = \mathbf{F}^{HLLC} + \mathbf{\Phi}^P, \tag{6.9}$$

$$\mathbf{\Phi}_p = (f-1)\frac{S_{own}S_{nei}}{S_{nei} - S_{own}}\frac{1}{1+|\tilde{M}|}\frac{\Delta p}{\tilde{c}^2}\begin{pmatrix} 1 \\ \tilde{\mathbf{u}} \\ \frac{1}{2}\tilde{u}^2 \end{pmatrix}. \tag{6.10}$$

$$f = \min\left(\frac{p_{own}}{p_{nei}}, \frac{p_{nei}}{p_{own}}\right)^3, \tag{6.11}$$

where the minimum of all cell faces is used for each cell.

$$p_{**} = \theta p_* + (1-\theta)\frac{p_{own} + p_{nei}}{2} \tag{6.12}$$

$$\theta = \min(M_*, 1) \tag{6.13}$$

$$p_{***} = f p_{**} + (1-f)p_*, \tag{6.14}$$

where p_{***} replaces p_{*K} in Eq. (6.7)

6.1.4 AUSM+

The AUSM+ scheme was developed by Luo et al. (2004) and constructs the fluxes based on the mass flux across the face.

$$\mathbf{F}^{AUSM+} = 0.5\left[M_* c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] - 0.5\left[|M_*| c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] + \mathbf{F}_p, \quad (6.15)$$

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_* \mathbf{n} \\ p_* \end{pmatrix}. \quad (6.16)$$

The star state variables are written as

$$c_* = 0.5(c_{own} + c_{nei}), \quad (6.17)$$

$$M_* = \mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei}, \quad (6.18)$$

and

$$P_* = \mathcal{P}^+_{5,own} p_{own} + \mathcal{P}^-_{5,nei} p_{nei}, \quad (6.19)$$

where

$$\mathcal{M}^\pm_1(M) = 0.5(M \pm |M|), \quad (6.20)$$

$$\mathcal{M}^\pm_2(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm 0.25(M \pm 1)^2 & \text{else} \end{cases}, \quad (6.21)$$

$$\mathcal{M}^\pm_4(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_2(M)\left[1 \mp 16\beta \mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases}, \quad (6.22)$$

$$\mathcal{P}^\pm_5(M) = \begin{cases} \frac{1}{M}\mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm \mathcal{M}^\pm_2(M)\left[(2 \mp M) - 16\alpha M \mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases}, \quad (6.23)$$

with $\alpha = 3/16$ and $\beta = 1/8$.

6.1.5 AUSM+up

The AUSM+ scheme extended to include a low speed correction by Liou (2006). The previous section is used to describe the high speed portion, however additional diffusion terms are added resulting in the following definitions of the mass and pressure fluxes

$$M_* = \mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei} + M_p, \tag{6.24}$$

and

$$P_* = \mathcal{P}^+_{5,own} p_{own} + \mathcal{P}^-_{5,nei} p_{nei} + P_u, \tag{6.25}$$

where

$$M_p = -\frac{K_p}{f_a} \max(1 - \sigma \bar{M}a^2, 0) \frac{p_{nei} - p_{own}}{\rho_* c_*^2}, \tag{6.26}$$

$$P_u = -K_u \mathcal{P}^+_{5,own} \mathcal{P}^-_{5,nei} (\rho_{own} + \rho_{nei}) f_a c_* (u_{nei} - u_{own}), \tag{6.27}$$

and $f_a = 1$, $\sigma = 1$, $K_p = 0.25$, $K_u = 0.75$, and α is redefined as $\frac{3}{16}(5f_a - 4)$. The interface density, ρ_*, is defined as $\frac{\rho_{own}+\rho_{nei}}{2}$, and the average Mach number is defined as.

$$\bar{M}a^2 = \sqrt{\frac{u_{own}^2 + u_{nei}^2}{2c_*^2}}, \tag{6.28}$$

This correction is very important when considering blasts because the Mach numbers can become much less than unity, resulting in numerical checker-boarding.

6.1.6 AUSM+up (granular)

A modified version of the AUSM+up flux scheme has been developed by Houim and Oran (2016) and includes flux limiters in the event of packing. The flux of conservative variables, (mass, momentum, thermal energy, and granular energy) across a face is given by

$$\mathbf{F}_s^{AUSM+up} = \mathbf{F}_p + \dot{m} \begin{cases} \mathbf{U}_{own} & \text{if } \dot{m} > 0 \\ \mathbf{U}_{nei} & \text{if } \dot{m} \leq 0 \end{cases}, \tag{6.29}$$

where

$$\dot{m} = \mathcal{F} c_{1/2} M_{1/2} \begin{cases} \alpha_{own}\rho_{own} & \text{if } M_{1/2} > 0 \\ \alpha_{nei}\rho_{nei} & \text{if } M_{1/2} \leq 0 \end{cases}, \tag{6.30}$$

$$\mathcal{F} = \frac{(c_{1/2} - \epsilon)[1 + |M_{1/2}|(1 - \mathcal{G}/2)]}{2} \frac{\max(\alpha_{s,own}, \alpha_{s,nei})}{\alpha_{s,max}} [\alpha_{own}\rho_{own} - \alpha_{nei}\rho_{nei}], \tag{6.31}$$

and $\mathbf{F}_p = \langle 0, P_{tot,1/2}\mathbf{I}, 0, 0 \rangle$, with α_s being the total granular volume fraction (sum of all granular phases), $\alpha_{s,max}$ being the maximum volume fraction at which packing occurs, and ϵ being a small number (1e-10).
$\mathcal{G}$ is defined as

$$\mathcal{G} = \max(2(1 - \mathcal{D}\zeta^2), \tag{6.32}$$

ζ defined as

$$\zeta = \begin{cases} \frac{\alpha_s^M - \alpha_{s,crit}}{\alpha_{s,max} - \alpha_{s,crit}} & \text{if } \alpha_s^M > \alpha_{s,crit} \\ 0 & \text{if } \alpha_s^M < \alpha_{s,crit} \end{cases} \tag{6.33}$$

where α_s^M is the maximum granular volume fraction of the own/nei value, and $\alpha_{s,crit}$ being the minimum granular volume fraction that frictional stresses occur.
$c_{1/2}$ is defined as

$$c_{1/2} = \sqrt{\frac{\alpha_{own}\rho_{own}c_{own}^2 + \alpha_{nei}\rho_{nei}c_{nei}^2}{\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei}}} + \epsilon. \tag{6.34}$$

The split Mach number is defined as

$$\begin{aligned} M_{1/2} =& \mathcal{M}_{4,own}^{+} + \mathcal{M}_{4,nei}^{-} \\ & -2\frac{K_p}{f_a}\max(1 - \sigma\bar{M}^2, 0)\frac{P_{tot,nei}-, P_{tot,own}}{(\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei} + \epsilon)c_{1/2}^2}. \end{aligned} \tag{6.35}$$

where $M_4^{\pm}$ is calculate the same way as in the AUSM+ scheme and $f_a = 1$. K_p, and σ are defined as

$$K_p = 0.25 + 0.75(1 - \mathcal{G}/2), \tag{6.36}$$

$$\sigma = 0.75\sigma/2. \tag{6.37}$$

The pressure is defined as

$$\begin{aligned} P_{tot,1/2} = & \\ & -K_u f_a (c_{1/2} - \epsilon) \mathcal{P}^+_{5,own} \mathcal{P}^-_{5,nei} (\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei})(u_{nei} - u_{own}) \\ & + \mathcal{P}^+_{5,own} p_{tot,own} + \mathcal{P}^-_{5,nei} p_{tot,nei} \end{aligned} \tag{6.38}$$

with $\mathcal{P}^{\pm}_5$ being the same as in the AUSM+ scheme and

$$K_u = 0.75 + 0.25(1 - \mathcal{G}/2), \tag{6.39}$$

6.1.7 Tadmor/Kurganov

The Tadmor/Kurganov fluxes were developed by Kurganov and Tadmor (2000). The *rhoCentalFoam* solver (OpenCFD Ltd., 2018a; Greenshields et al., 2010) utilizes this scheme, which has been ported to the current framework. Two run-time selectable options (e.g. `Kurganov` or `Tadmor`) are available; both follow the same general procedure with a slight difference in the calculation of coefficients.

The Kurganov scheme defines

$$a^+ = \max(\max(u_{own} + c_{own}, u_{nei} + c_{nei}), 0), \tag{6.40}$$

$$a^- = \min(\min(u_{own} - c_{own}, u_{nei} - c_{nei}), 0), \tag{6.41}$$

$$a_{own} = \frac{a^+}{a^+ + a^-}, \tag{6.42}$$

$$a_{nei} = \frac{a^-}{a^+ + a^-}, \tag{6.43}$$

and

$$a = \frac{a^- a^+}{a^+ + a^-}. \tag{6.44}$$

The Tadmor scheme defines $a_{own} = a_{nei} = 0.5$ and $a = max(|a^-|, |a^+|)$.

The volumetric fluxes are written

$$\phi_{own} = u_{own} a_{own} - a, \tag{6.45}$$

and

$$\phi_{nei} = u_{nei} a_{nei} + a. \tag{6.46}$$

The resulting fluxes are

$$\mathbf{F}^{KT} = (\phi_{own} \mathbf{U}_{own} + \phi_{nei} \mathbf{U}_{nei}) + \mathbf{F}_p, \tag{6.47}$$

with

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (a_{own} p_{own} + a_{nei} p_{nei}) \mathbf{n} \\ a(p_{own} - p_{nei}) \end{pmatrix}. \tag{6.48}$$

6.2 Riemann advection scheme

In order to allow for consistency between the conserved quantities (mass, momentum, and energy), and additional transport quantities such as turbulence fields or species mass fractions, an additional advection scheme using the Riemann fluxes has been added. This makes use of the quantities calculated in transport of the conserved variable which then allows for the use of higher order interpolation schemes to be used for additional scalar transport. This is a major advantage over traditional methods which are generally limited to upwinding for additional transport due to presence of strong gradients.

```
divSchemes
{
    default                          none;
    div(alphaRhoPhi.tnt,lambda.tnt)  Riemann;
    div(alphaRhoPhi.tnt,c.tnt)       Riemann;
}

interpolationSchemes
```

```
{
    default cubic;
    ...

    reconstruct(lambda.tnt)  vanLeer;
    reconstruct(c.tnt)       vanLeer;
}
```

In the interpolation schemes sub-dictionary, the limiters should be specified for each field using the Riemann fluxes, but by default the limiter specified by "reconstruct(rho)" will be used.

NOTE: One limitation of using the Riemann fluxes is that, because an explicit advection term for the transported quantities is used, when very stiff implicit terms are present (such as production term in turbulence models), the advection term can become negligible leading to no advection occurring. For this reason, implicit divergence schemes are still recommended for the advection of turbulence quantities and other equations with large implicit source terms.

6.3 Time integration

Higher order time integration has been added allowing for fully third order accurate solutions to the evolution equations. This is a major benefit in comparison to standard OpenFOAM time integration which is at most second order due to the limitation on the implicit time evolution. This functionality applies to both conservative quantities such as mass, momentum, and energy, as well as the activation and afterburn models. The transport of turbulent quantities are still limited by the standard time integration schemes in OpenFOAM. Below are the currently available time integration schemes and the mathematical steps associated with them. The superscript n denotes the state at the beginning of the current time step, (i) denotes the i-th step, and $n+1$ denotes final state. The time integration scheme is specified in the `fvSchemes` dictionary using

```
ddtSchemes
{
    default Euler;
    timeIntegrator RK2SSP 2; // Two stage, second order accurate
}
```

NOTE: The strong stability preserving (SSP) methods offer several different variation that allow for a greater number of steps to be used to add additional stability to the solution. This is specified by adding the number of requested steps after the SSP scheme. If a number is provided that is higher or lower than the available number of steps, the scheme will used the closest number of steps.

6.3.1 Euler

Standard first order time integration

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.49}$$

6.3.2 RK2

Standard second-order Runge-Kutta method (mid point):

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.50}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{6.51}$$

6.3.3 RK2-SSP (RK2SSP)

Second-order, strong-stability-preserving Runge-Kutta method (Spiteri and Ruuth, 2002):
Two stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.52}$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\left(\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right). \tag{6.53}$$

Three stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.54}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{6.55}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\mathbf{U}^n + \frac{2}{3}\mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)}. \tag{6.56}$$

Four stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.57}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{6.58}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^{(2)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{6.59}$$

$$\mathbf{U}^{n+1} = \frac{1}{4}\mathbf{U}^n + \frac{3}{4}\mathbf{U}^{(3)} - \frac{1}{4}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(3)}. \tag{6.60}$$

6.3.4 RK3-SSP (RK3SSP)

Third-order, strong-stability-preserving Runge-Kutta method (Spiteri and Ruuth, 2002):
Three stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.61}$$

$$\mathbf{U}^{(2)} = \frac{1}{4}\left(3\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{6.62}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\left(3\mathbf{U}^n + 2\mathbf{U}^{(2)} - 2\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{6.63}$$

Four stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.64}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{6.65}$$

$$\mathbf{U}^{(3)} = \frac{2}{3}\mathbf{U}^n + \frac{1}{3} - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{6.66}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^{(3)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(3)}. \tag{6.67}$$

6.3.5 RK4

Standard fourth-order Runge-Kutta method:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{6.68}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{6.69}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{6.70}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \left(\mathbf{F}^n + 2\mathbf{F}^{(1)} + 2\mathbf{F}^{(2)} + \mathbf{F}^{(3)}\right) \tag{6.71}$$

6.3.6 RK4-SSP (RK4SSP)

Fourth-order, strong-stability-preserving Runge-Kutta method (Spiteri and Ruuth, 2002):

$$\mathbf{U}^{(i)} = \sum_{k=0}^{i-1} \left(a_{ik}\mathbf{U}^{(k)} + \Delta t \beta_{ik} \boldsymbol{\nabla} \cdot \mathbf{F}^{(k)}\right) \tag{6.72}$$

Table 1: a_{ik}

0	1			
1	$\frac{649}{1600}$	$\frac{951}{1600}$		
2	$\frac{53989}{2500000}$	$\frac{4806213}{20000000}$	$\frac{23619}{32000}$	
3	$\frac{1}{5}$	$\frac{6127}{30000}$	$\frac{7873}{30000}$	$\frac{1}{3}$
	0	1	2	3

Table 2: β_{ik}

0	1			
1	$\frac{-10890423}{25193600}$	$\frac{5000}{7873}$		
2	$\frac{-102261}{5000000}$	$\frac{-5121}{20000}$	$\frac{7873}{10000}$	
3	$\frac{1}{10}$	$\frac{1}{6}$	0	$\frac{1}{6}$
	0	1	2	3

7 Adaptive Mesh Refinement (AMR)

The dynamic mesh classes have been incorporated from the github repository SOFTX_2018_143 and is based on the work of Rettenmaier et al. (2019). This library introduces 2-D, 2-D axisymmetic, and 3-D (present in OpenFOAM-7) all with the option to include dynamic load balancing. The original work (*dynamicRefineBalanceFvMesh* and *dynamicRefineFvMesh*) has been used a the base for the *adaptiveFvMesh* class. This mesh type has been further developed, specifically for compressible flows, and include modifications for improved stability with meshes created with *snappyHexMesh*.

Because the detonation of explosives may not occur when the simulation starts, the option to begin unrefinement at a designated time (*beginUnrefine*) has been added. This allows the mesh to remain refined in specific regions even though the error is small initially.

Additionally, based on *adaptiveFvMesh* is the *movingAdaptiveFvMesh* class which has all of the same features, but included the ability to use mesh motion.

NOTE: Whilst AMR may be used with meshes created using *snappyHexMesh*, regions can only be cut at level 0 when using either 2-D or axisymmetric refinement due to the fact that *snappyHexMesh* uses a 3-D refinement, regardless of the number of solution directions. If refinement is used, the simulation will crash when the mesh is refined in these areas.

NOTE: Meshes that have been "snapped" with *snappyHexMesh* may work, however crashes may occur. Use at your own risk.

Entry	Description
protectedPatches	List of patch names that will not be refined or unrefined (default is none)
beginUnrefine	Time when unrefinement can begin (default is 0 s)

7.1 Error estimators

In order to determine the cells that can be refined or unrefined, error estimators have been added. Generally, this is achieved by looking for a derivative or difference of a specified fields across all the faces a cell and it neighbours. The maximum value of all face errors of a given cell is taken as the cell center-based value, and is used to define the cells that should be refined.

Entry	Description
errorEstimator	Method used to calculate the error
beginUnrefine	Time that unrefinement can begin (0 by default)
refineInterval	Number of timesteps between refinement/unrefinement
field	Field used to deterermine refinement/unrefinement (error if an errorEstimator is used)
lowerRefineLevel	Error that mesh is refined at
unrefineLevel	Error that mesh is unrefined
upperRefineLevel	Error that refinement is stopped (default is great)
upperUnrefineLevel	Error that unrefinement begins to occur again (default is great)
nBufferLayers	Number of cells outside the refinement area that are refined
maxRefinement	Maximum number of refinement levels
maxCells	Maximum number of cells before refinement stops (maxLabel by default)
correctFluxes	Optional list of ways to correct fluxes (not used with `blastFoam`)
dumpLevel	Is the refinement level written

7.1.1 fieldValue

The error is set to the value of the given field

$$e_x = x \tag{7.1}$$

where x is a field chosen by the user, and can be any scalar field stored within the simulation.

Entry	Description
field	Error field

7.1.2 delta

A basic method to detect sharp changes across faces uses a difference approach, where the error is defined by

$$e_\delta = |x_{own} - x_{nei}| \tag{7.2}$$

where x is a field chosen by the user, and can be any scalar field stored within the simulation.

Entry	Description
field	Field used to compute error

7.1.3 scaledDelta

A basic method to detect sharp changes across faces uses a scaled difference approach, where the error is defined by

$$e_{|\delta|} = \frac{|x_{own} - x_{nei}|}{\max(\min(x_{own}, x_{nei}), minValue)} \tag{7.3}$$

where x is a field chosen by the user, and can be any field stored within the simulation. The minValue can be specified by the user and is the minimum value that the error will be scaled by.

Entry	Description
field	Field used to compute error
minValue	minimum value to scale difference by (default is small)

7.1.4 Density gradient (densityGradient)

A modified density gradient was defined by Zheng et al. (2011), and uses an error defined by

$$e_{\boldsymbol{\nabla}\rho} = \max\left(\frac{|(\boldsymbol{\nabla} l_n \rho)_C - (\boldsymbol{\nabla} l_n \rho)_{own}|}{\alpha_f \rho_C / dl + |\boldsymbol{\nabla} l_n \rho)_{own}|}, \frac{|(\boldsymbol{\nabla} l_n \rho)_C - (\boldsymbol{\nabla} l_n \rho)_{nei}|}{\alpha_f \rho_C / dl + |\boldsymbol{\nabla} l_n \rho)_{nei}|}\right)_f, \tag{7.4}$$

with

$$(\boldsymbol{\nabla} l_n \rho)_C = \frac{\rho_{nei} - \rho_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|}, \tag{7.5}$$

$$(\boldsymbol{\nabla} l_n \rho)_{nei,own} = (\boldsymbol{\nabla} \rho)_{nei,own} \frac{\mathbf{r}_{nei} - \mathbf{r}_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|}, \tag{7.6}$$

$\mathrm{r}_{own,nei}$ is the surface normal vector, and $dl = |\mathbf{r}_{nei} - \mathbf{r}_{own}$, is the distance between cell centers. This error estimator uses the total density.

7.1.5 Lohner

A final error estimation method based on Löhner (1987) is also available. It uses a scaled Laplacian to ensure an error bounded between 0 and 1 and indicates the smoothness of the solution. Due to the fact that the method was initially developed for a Cartesian grid, some modifications were required. Namely, the face value of the tracked field is interpolated using a cubic interpolation method. This allows non-zero smoothness across the faces while still allowing for the method to be used on a non-Cartesian grid.

$$e_{Lohner} = \frac{|x_{own} - 2x_f + x_{nei}|}{|x_{own} - x_f| + |x_{nei} - x_f| + \varepsilon(|x_{own}| + 2|x_f| + |x_{nei}|)}, \tag{7.7}$$

where ε is a coefficient which is problem dependent, and x is any scalar field saved within the simulation.

Entry	Description
field	Field name used to compute error
ε	Scaling coefficient

7.1.6 multiComponent

The multicomponent error estimator allows for multiple errors to be used to define which cells should be refined. Any of the previous methods can be used. A list of names for the error estimators need to be provided, which must correspond to a dictionary with the necessary entries included. The maximum of error at each cell is used for refinement.

```
// BlastFoam specific AMR mesh type
dynamicFvMesh   adaptiveFvMesh;

// Error is calculated using the density gradient
errorEstimator  multicomponent;
estimators (density pressure volumeFraction);
density
{
    errorEstimator densityGradient;
    lowerRefineLevel 0.01;
```

```
    unrefineLevel    0.01;
}
pressure
{
    errorEstimator  scaledDelta;
    deltaField      p;
    minValue        1e3;

    lowerRefineLevel 0.1;
    unrefineLevel    0.01;
}
volumeFraction
{
    errorEstimate   fieldValue;
    fieldName       alpha.tnt;

    //- Only refine when 0.1 < alpha.tnt < 0.6
    lowerRefineLevel 0.1;
    unrefineLevel    0.05;

    upperRefineLevel 0.6;
    upperUnrefineLevel 0.7;
}
```

Entry	Description
estimators	List of error estimators to be used

7.2 Dynamic Load Balancing

The ability to use dynamic load balancing is available for 2-D, 2-D axisymmetic, and 3-D meshes. Dynamic load balancing can greatly reduce the computational time required by balancing the cells equally across all processors. However, there are known limitations: due to the way the OpenFOAM transfers data between processors, the MPI buffer limit can be exceeded for large cases, leading to the simulation crashing. There are also known problems in the transferring of data for calculate boundary conditions. The fields

used within the standard `blastFoam` solver have addressed this, but if *adaptiveFvMesh* is used with a different solver, its stability with balancing is not guaranteed.

Entry	Description
balance	Is the mesh balance (default is yes)
balanceInterval	How many refine/unrefine steps occur between balances (default is 1)
allowableImbalance	Maximum deviation for average number of cells on a processor (default is 0.2)
method	Method used to balance the mesh (default is method used in `decomposeParDict`)

7.3 Example

An example of the optional `dynamicMeshDict` for use with *adaptiveFvMesh* class

```
// BlastFoam specific AMR mesh type
dynamicFvMesh   adaptiveFvMesh;

// Error is calculated using the density gradient
errorEstimator  densityGradient;

loadBalance // Can only be used with 3-D geometries
{
    // Is the mesh dynamically balanced
    balance yes;

    // How many refinement steps between rebalancing
    balanceInterval 20;

    // Allowable difference between average load
    // and max load
    allowableImbalance 0.2;

    // Decomposition method
    method scotch;
```

```
}

// How often to refine
refineInterval  1;

// When to begin unrefinement
beginUnrefine 1e-5;

// Refine field in between lower..upper
lowerRefineLevel 1e-2;

// If value < unrefineLevel unrefine
unrefineLevel   1e-2;

// Number of cells between level
nBufferLayers   1;

// Refine cells only up to maxRefinement levels
maxRefinement   4;

// Write the refinement level as a volScalarField
dumpLevel       true;
```

8 Coupling to preCICE's OpenFOAM adapter

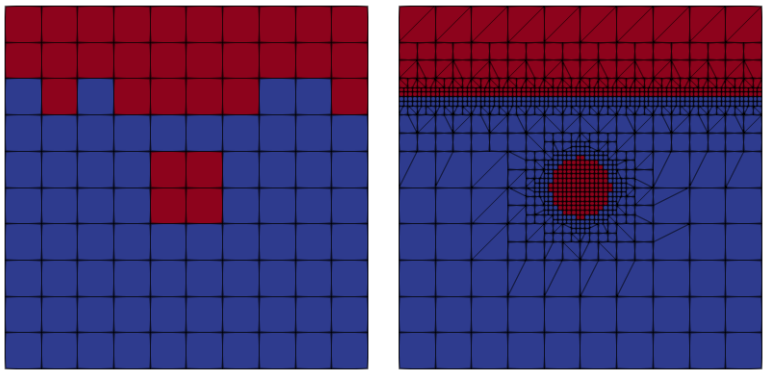
With the addition of moving meshes, the ability to couple `blastFoam` to an external solver through preCICE is possible. There is no special setup and is ready to be used upon install. There are no limitations for the standard *blastFoam* moving mesh classes. However, when using the *movingAdaptiveFvMesh*, the coupled boundary cannot change within the simulation. For this reason balancing is not allowed. The optional *protetedPatches* keyword must be used to specify boundaries which cannot be refined.

9 Utilities

Listed below are the utilities that have been developed for *blastFoam* to simplify case setup and post-processing.

9.1 *setRefinedFields*

This is a modified version of the *setFields* utility in the standard OpenFOAM that includes the ability to refine the mesh based on the fields that are set. This is extremely beneficial when a small volume needs to be set within a set of very large computational cells. The same functionality presented in `adaptiveFvMesh` is present allowing for 2D and 3D geometries to be set. The utility works by setting the default field values, setting the cell or face sets, then checking the difference in a specified set of fields across all faces. Each region must have a level defined inside where the mesh is refined up to this level. The cells inside the region can also be refined using the *refineInternal* keyword. The default field values for any fields being set must also be defined so that the region is updated at each iteration. The utility will iterate until the mesh is no longer changing, or the maximum number of iterations is reached. A comparison using the standard *setFields* and setRefinedFields can be seen in Fig. 1.



(a) Unrefined initial field (b) Refined initial field

Figure 1: Comparison of initial fields using *setFields* and *setRefinedFields*

By default no error estimator will be used, so any difference over a face will be resolved, however by specify an error estimator, the refinement will

be determined by the spacial derivatives of the fields. This can be useful if starting from an existing solution (see Sec. 9.6). If this option is used the *maxLevel* must be specified if no refinement regions are specified. If a refinement region is specified as well, the max refinement level will be taken as the maximum level provided in the regions.
By default only fields that are specified within *setFieldsDict* will be modified, meaning that all other fields must use uniform field values to be read. The `-updateAll` command line option will read in all fields present in the current time step and update the sizes of the fields which is necessary is non-uniform fields will be used.
In addition to the standard shapes that can be set with *setFields* the option to set a list of sphere or cylinders has been added. These can be used with the types `sphereListToCell` and `cylinderListToCell` respectively. Theses options take the following parameter: radii and centres for sphere, and radii, p1s, and p2s for cylinders.

Entry	Description
fields	Field names that are used to compute error and refine mesh
level	Level of each cell set
refineInternal	Are the cells inside each region refined
refineFaces	Are the owner and neighbor cells of faces refine
nBufferLayer	Number of cells between level boundaries
backup	Dictionary used if a cell set does not contain any cells
maxCells	Maximum number of cells (default is the max interger)
maxIter	Maximum number of iterations (default is 2*max(level)
force3D	Used 3D refinement so that standard OpenFOAM solvers can be used (default is no)
errorEstimator	Type of error estimator used (default none)
debug	Write intermediate step with error and max cell level (default is no)

An example `setFieldsDict` is listed below

```
fields (alpha.air);
nBufferLayers 1;

defaultFieldValues
(
    volVectorFieldValue U ( 0 0 0 )
```

```
    volScalarFieldValue p 1.1e5
    volScalarFieldValue alpha.air 0
    volScalarFieldValue rho.air 1
    volScalarFieldValue rho.water 1000
);


regions
(
    cylinderToCell
    {
        p1 (0 0 -1);
        p2 (0 0 1);
        radius 0.1;
        level 3;
        refineInternal yes;
        backup
        {
            p1 (0 0 -1);
            p2 (0 0 1);
            radius 0.2;
        }

        fieldValues
        (
            volScalarFieldValue p 9.12e8
            volScalarFieldValue alpha.air 1
            volScalarFieldValue rho.air 1270
        );
    }

    boxToCell
    {
        boxes ((-0.6 0.3 -1) (0.6 0.6 1));

        fieldValues
        (
            volScalarFieldValue alpha.air 1
```

```
        );
    }
    patchToFace
    {
        name walls;
        level 2;
        refineFaces yes;
    }
);
```

NOTE: Because OpenFOAM does not decompose both the time-based mesh (e.g. *0, 0.001, ...*) and the base mesh (located in *constant*), where both are required to use the mesh modified by *setRefinedFields*, if it is run in parallel, *setRefinedFields* must be run in parallel ***after*** the case has been decomposed.

9.2 *blastToVTK*

In order to save on space, it can often be preferable to write VTK files at timesteps between the specified output times. Due to the way that OpenFOAM writes this data, it can become tedious to view the time series of each data set. In order to make this process simpler, the utility *blastToVTK* will create symbolic links (or create copies) within the VTK directory in the case path so that that the collections of VTK files can be viewed in Paraview in a time series. The real time values are not present due to the fact that they are not written when the VTK files are, instead they are listed using integer indexes. The files will be in the correct order when viewing the collection.

By default only new VTK files will have symbolic links created, however, if the user would like to make hard copies of the files, the "-hardCopy" option can be used (this may be required on Windows-based platforms). If `blastToVTK` needs to be rerun, the "-force" option should be used to replace the previous links.

9.3 *createVTKTimeSeries*

This utility simplifies the previous utility by simply creating a *.vtu file which provided the path each time step of a vtk sampled surface and allows

viewing of the time series with the time in *paraview*. Only one additional file is created for each sampled field in the VTK directory.

9.4 *calculateImpulse*

This is a simple utility that can be used to calculate the impulse using a pressure probe. The utility is run

```
calculateImpuse -probeDir pProbes -pRef 101325
```

where "-probeDir" is the name of the probe as listed in the `controlDict` and "-pRef" is the reference pressure. Additionally the "-name" option can be used to specify the name of the pressure field if some other name is used. This utility will integrate all probes within the pressure file using the time steps listed.

9.5 *initializeAtomsphere*

The *initializeAtmosphere* utility is used to initialize stable pressure and density fields so that the hydrostatic pressure gradient and gravitational acceleration force balance each other. Currently there are two available models, hydrostatic and tabulated. The hydrostatic model uses a constant density to initialize a stable pressure gradient. The tabulated model reads a pressure and temperature file and uses ideal gas relations to determine the resulting density field.

9.6 *rotateFields*

The *rotateFields* utility is can be used to map the fields of a 1-D or 2-D solution to that of a 2-D or 3-D solution, where the source and target meshes are used to determine the rotation. The rotation of vector and tensor fields are mapped using the local rotation tensor to ensure that the correct radial components are used. There is no dictionary used to read inputs to the utility, instead the command line options are

Entry	Description
-sourceTime	Time in the source case to map (default is startFrom time in source case)
-sourceRegion	Region of the source case to map (only used for multi-region)
-targetRegion	Region of the target case to map to (only used for multi-region)
-extend	If a cell is outside of the source range, mapo to the closest cell (default is no)
-maxR	Cut off radius to map (default is a big number)
-centre	Center of rotation on the source mesh (default is (0 0 0))
-fields	List of fields to map (default is all present)

NOTE: Currently symmetric and spherical tensors cannot be mapped since the rotation could change the fields to non-symmetric or non-spherical, and therefore change the type
NOTE: Currently mapping from decomposed cases is not supported, but mapping to a decomposed case is.

10 Function objects

In addition to the standard OpenFOAM `functionObjects`, several additional blast specific function object have been added. These can be included using the following `controlDict` entry

NOTE: Using function objects not standard to *blastFoam* may result in warning messages when a simulation is started due to duplicate entries in run time selection tables.

```
functions
{
    impulse
    {
        type    impulse;
        // writeTime - Writes every output time
        // timeStep - Number of time steps
        // adjustableRunTime - Specified output interval
        writeControl    writeTime;

        // Interval between writes
        writeInterval 1;

        // Reference pressure
        pRef 101325;

        ...
    }
}
```

10.1 Mach number (singlePhaseMachNo)

The *singlePhaseMachNo* replaces the standard machNo available in OpenFOAM by using the speed of sound calculated from the *blastFoam* specific thermodynamic models.

Entry	Description
phaseName	Name of the phase (default is none)

10.2 Speed of sound (speedOfSound)

Similar to the *singlePhaseMachNo*, the *speedOfSound* function object used the calculated speed of sound from the thermodynamic model and writes the field. The field can also be saved within the simulation to allow for sampling with probes or other function objects.

Entry	Description
phaseName	Name of the phase (default is none)
store	Is the field saved in the simulation (default in no)

10.3 dynamicPressure

It can be useful in blast simulations to track the dynamic pressure of a phase defined as

$$\mathbf{p}_{dyn} = \frac{1}{2}\rho_i \mathbf{U}|\mathbf{U}|. \tag{10.1}$$

The function object returns the vector of dynamic pressure, where the magnitude of this output is the standard dynamic pressure. The field can also be saved within the simulation so that additional sampling can be conducted (i.e. probes for time series sampling).

Entry	Description
rhoName	Name of the density field (default is rho)
UName	Name of the velocity field (default is U)
store	Is the field stored (default is no)

10.4 Max field value over time (fieldMax)

The maximum value at every cell in a domain can be output, allowing for quantities such as maximum impulse and peak pressure to be visualized. This can be done for any field currently stored within the simulation. These fields are automatically saved within the calculation.

Entry	Description
fields	Name of the fields (i.e (rho U p impulse))

10.5 impulse

This function object calculates the impulse in every cell by integrating the overpressure over time using the specified pressure field and reference pressure. The impulse field is automatically saved to allow additional sampling by default.

Entry	Description
pName	Name of the pressure field (default is p)
pRef	Reference pressure value [Pa]

10.6 overpressure

This function object calculates and writes the overpressure (e.g. the pressure relative to a given reference pressure, usually ambient) given a pressure field and reference pressure. The field can also be saved to allow for additional sampling.

Entry	Description
pName	Name of the pressure field (default is p)
pRef	Reference pressure [Pa]
store	Is the field saved (default is no)

10.7 Time of arrival (timeOfArrival)

This function object tracks the maximum pressure and records the time when the maximum pressure is reached.

Entry	Description
pName	Name of the pressure field (default is p)

10.8 writeTimeList

This function object writes all fields and mesh at a list of given times. This is useful when write times are needed that do not line up with any realistic write interval.

Entry	Description
times	List of times to write

10.9 conservedQuantities

The volume integrated value of values can be computed and printed in the output of the solver. The initial, current, total difference, and relative difference is printed. Any field can be tracked by specifying the names of the fields in the dictionary.

Entry	Description
fields	Fields to track

NOTE: Currently the removal and injection of values is not considered.

10.10 Tracer particles (tracerParticles)

The use of non-inertial particles can be added to any simulation, and the evolution of tracer particles is determined by the local quantities (velocity, density, etc.). The `kinematicCloudProperites` file must be included to provide the inputs of the tracer particles. Below is an example input

```
solution
{
    active          true;
    transient       true;
    coupled         false;
    cellValueSourceCorrection off;
    sourceTerms
    {
        schemes
        {
            U               semiImplicit 1;
        }
    }
    integrationSchemes
    {
        U               Euler;
    }
    interpolationSchemes
    {
```

```
        rho             cell;
        U               cell;
        thermo:mu       cell;
    }
}
constantProperties
{
    rho0            1;
}
subModels
{
    particleForces
    {
        instantDrag;
    }
    injectionModels
    {
        model1
        {
            type            patchInjection;
            patchName       inlet;
            parcelBasisType  mass; // fixed;

            SOI             0;    // Start time of injection
            duration        0.1;

            U0              (200 0 0);
            massTotal       1;
            parcelsPerSecond 1e4;
            flowRateProfile  constant 1;

            sizeDistribution
            {
                type        fixedValue;
                fixedValueDistribution
                {
                    value 1e-3;
                }
```

```
            }
        }
    }
    dispersionModel none;
    patchInteractionModel localInteraction;
    localInteractionCoeffs
    {
        patches
        (
            outlet
            {
                type escape;
            }
            inlet
            {
                type none;
            }
            aerofoil
            {
                type rebound;  // For Un = normal speed; Ut = tangential speed
                e    1;        // e = (Un_reflected / Un_incident)
                mu   0;        // mu = [1 - (Ut_reflected / Ut_incident)]
            }
        );
    }
    stochasticCollisionModel none;
    collisionModel none;
}
```

10.11 probes

A modified version of the standard OpenFOAM probe utility is also available where the basic functionality remains the same, with several key additions. First the ability to append to exiting probe files is now possible allowing for a single probe file to be used even if a simulation is restarted from a later time. Second, probes that are not found within the mesh can be moved to the nearest face so they can produce valid results. This can be very useful if a probe is needed on a curved face. Lastly, the warning of probes existing

on multiple processors has been removed by selecting a single processor on which to probe.
NOTE: In order to use any of the "point" based interpolation schemes, the Synthetik fork of OpenFOAM-7 must be used.
NOTE: When refinement and unrefinement occurs the probe locations will "jump" to the nearest cell resulting in fluctuations when plotted. Using the cellPoint interpolationScheme will reduce this, but is cannot be eliminated.

Entry	Description
probeLocations	List of locations to probe
fields	Fields to probe at the given points
append	Are probes appended to and existing file (default no)
adjustLocations	Can probes be moved into the mesh (default no)
interpolationScheme	Method to interpolate probe. Options are cell, cellPoint, cellPointFace, cellPatchConstrained, cellPointWallModified, and pointMVC (default is cell)

11 Solvers

blastFoam offers a wider variety of solver for solver highly compressible flows. A short description of each is given below.

11.1 blastFoam

The blastFoam solver is the standard solver and has a variety of uses, including single phase, two phase and multiphase flows, where the phases and thermodynamic models are specified in the `phaseProperties`. When a list of phases is provided (and the list has more than one entry), the two phase or multiphase solver is selected. If the key word phases is not specified, then the single phase solver is selected. All number of phases use the *blastFoam* specific equations of states (see Section 4). This solver has been specifically designed for simulating detonating explosive materials and has the option to use any of the models presented.

11.2 blastEulerFoam

The *blastEulerFoam* solver is the Euler-Euler (two/multifluid) variant of blastFoam. Currently only fluid-solid simulations are possible, however, the addition of fluid-fluid simulations is under development. This solver is based on the work of Houim and Oran (2016) and Lai et al. (2018). One major advantage over the standard *reactingEulerFoam* solver is that any number of particle phases can be used. Currently there are three phase models that can be used: fluid, multicomponent, and granular. The fluid phase model is used to describe a single component fluid (such as air). The multicomponent phase model is used to describe a fluid phase consisting of multiple species or components (air and c4, N2 and He, etc.). The multiphase phase model is used to describe a fluid phase consisting of multiple phases, each described by a unique equation of state. This allows simulations of standard detonation problems with the addition of particles. The granular phase model is used to describe a solid phase. Both the fluid, multicomponent, and multiphase models can use any of the fluid thermodynamic models, while the granular model can use any of the solid thermodynamic models. The models describing interactions between fluids are described in Sec. 2.3 and Sec. 2.4

11.3 blastXiFoam (Experimental)

In addition to the standard blastFoam solver, an additional solver has been added to solve combustion as an extension of the standard OpenFOAM XiFoam solver. The only difference is that the flux schemes presented earlier are used to transport the conserved quantities. This allows for a more accurate description of combusting, highly compressible flows (i.e. deflagration to detonation transition). It is recommended to use the *adibaticFlameT* utility to calculate the model coefficients used in the required *thermophysicalProperties* dictionary.

11.4 blastReactingFoam (Experimental)

blastReactingFoam uses the standard OpenFOAM thermodynamic classes, and can be used to solve multi-specie systems that include reactions. The use of combustion models are also available. The `thermophysicalProperties` dictionary must be included under the constant dictionary along with the relevant dictionaries (`combustionProperties` or `chemistyProperties`) and initial value files (mass fractions). The conserved quantities (mass, momentum, and energy) as well as the mass fractions are all transported using the flux schemes presented herein.

11.5 blastMultiRegionFoam (Experimental)

The *blastMultiRegionFoam* solver is derived from the *chtMultiRegionFoam* available in OpenFOAM. The major differences are in the use of ODE time integration and Riemann fluxes. The thermal transport in solids is still treated implicitly through a matrix solution.

11.6 blastFSIFoam (Experimental)

The *blastFSIFoam* used the standard *blastFoam* solver to solve the fluid phase, and the OpenFOAM standard *solidDisplacementFoam* solver to handle the solid deformation. The solid phase uses a moving mesh, while the solid phase is solved on a stationary mesh. In order to correctly transfer the information between the patches, the *mappedMovingWall* patch type must be used. This is done by modifying the type within `constant/region*/polyMesh/boundary` for each region (See any of the *blastFSIFoam* tutorials or validation cases).

NOTE: Dynamic load balancing cannot currently be used.

NOTE: This solver is still in development and is not yet stable. Please use with caution.

11.7 blastParcelFoam (Experimental)

blastParcelFoam makes use of the standard OpenFOAM Lagrangian classes to solve a coupled Eulerian fluid and collection of Lagragian parcels. This includes interaction terms like drag, heat transfer, and particle collisions. Not all of the OpenFOAM standard Lagrangian options are available due to the use of a different thermodynamic model.

11.8 Quadrature-based moment method (QBMM) solvers (Experimental)

BlastFoam has the ability to couple with OpenQBMM to solve compressible flows with number density based transport. Three options currently exist, *blastPbeTransportFoam* for uncoupled, univariate moment transport; *blastUncoupledVdfTransportFoam* for uncoupled, multivariate (specifically velocity and size velocity) moment transport; and *blastVdfTransportFoam* for coupled multivariate (specifically velocity and size velocity) moment transport. All of these solvers include source terms to account for size changes due to aggregation, breakup, growth (or evaporation), and nucleation. Please refer to Yuan and Fox (2011); Madadi-Kandjani and Passalacqua (2015); Passalacqua et al. (2018) for details on the implementation and usage of quadrature methods.

12 Example Cases

Two validation case and one tutorial case are presented to show the general case setup and run procedure. The first case is a single phase, double mach reflection problem. The second is a simple 1-D shock tube consisting of air and water. The third case is a 2-D, three-phase case with two different detonating phases in which the first explosive material is used to detonate the second charge using the pressure based activation model. The final case describes a burried charge of c4 in a sand. Only the new or modified files will be shown, but the other required files such as `blockMeshDict` and the initial conditions can be found in the case directories (`validation/blastFoam/doubleMachReflection`, `validation/blastFoam/shockTube_twoFluid`, `tutorials/blastFoam/twoChargeDetonation`) and `tutorials/blastEulerFoam/mine`).

12.1 Double mach reflection

The double mach reflection problem is a complex case based on Woodward and Colella (1984) which is meant to demonstrate the capabilities of the single phase solver within *blastFoam*. Coded boundary conditions for velocity, pressure and density are used to correctly describe the shock behavior at the top boundary. The case uses the *singlePhaseCompressibleSystem* along with the RK2-SSP ODE time integrator and HLLC flux scheme.

The challenge in setting up the double mach reflection is due to the fact that the boundary conditions must be set so that the fields on the top boundary follow the correct upstream and downstream shock conditions which are

$$\rho_R = 1.4 \text{ kg/m}^3,\ p_L = 1 \text{ Pa},\ T_R = 1 \text{ [K]},\ \mathbf{U}_R = (0,\ 0,\ 0) \text{ m/s},$$

$$\rho_L = 8 \text{ kg/m}^3\ p_L = 116.5 \text{ Pa},\ T_R = 20.3887 \text{ [K]}\ \mathbf{U}_L = (7.1447,\ -4.125,\ 0) \text{ m/s}.$$

These conditions change based on the shock location, leading to a modified boundary condition for density, pressure, temperature, and velocity which uses a codedFixedValue, i.e. the initial density field is defined using

```
dimensions      [1 -3 0 0 0 0 0];

internalField   uniform 1.4;
```

```
boundaryField
{
    inlet1
    {
        type            fixedValue;
        value           $internalField;
    }
    inlet2
    {
        type            fixedValue;
        value           $internalField;
    }
    outlet
    {
        type            codedFixedValue;
        value           $internalField;
        name            shockRho;
        code
        #{
            // Shock location (x)
            scalar xS =
                db().time().value()*11.2799
              + 0.74402;
            scalarField x =
                this->patch().Cf().component(0);

            operator==
            (
                pos(x - xS)*1.4
              + neg0(x-xS)*8.0
            );
        #};
    }
    walls
    {
        type            zeroGradient;
    }
```

```
    defaultFaces
    {
        type            empty;
    }
}
```

The velocity and temperature fields are initialized using the upstream and downstream values with the same evolution equations. The pressure field used zeroGradient boundary conditions. The internal and boundaries are set using *setFields* with the `setFieldsDict` defined as

```
defaultFieldValues
(
    volVectorFieldValue U (7.1447 -4.125 0)
    volScalarFieldValue T 20.3887
    volScalarFieldValue p 116.5
    volScalarFieldValue rho 8.0
);

regions
(
    rotatedBoxToCell
    {
        origin (0.16667 0 -1);
        i (8.66025 -5.0 0);
        j (5.0 8.66025 0 );
        k (0 0 2);
        fieldValues
        (
            volScalarFieldValue p 1
            volScalarFieldValue T 1
            volVectorFieldValue U (0 0 0)
            volScalarFieldValue rho 1.4
        );
    }
    boxToFace
    {
        box (0.74402 0.999 -1) (5 1.001 1);
```

```
        fieldValues
        (
            volScalarFieldValue p 1
            volScalarFieldValue T 1
            volVectorFieldValue U (0 0 0)
            volScalarFieldValue rho 1.4
        );
    }
);
```

The `phaseProperties` file is

```
mixture
{
    type          basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    specie
    {
        molWeight   11640.3; // normalizes pressure
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       1.4;
    }
    thermo
    {
        Cv          1.785714286;
        Hf          0.0;
```

```
    }
}
```

The case is then initialized and run using the following commands:

```
blockMesh
setFields
blastFoam
```

The final results can be seen below.

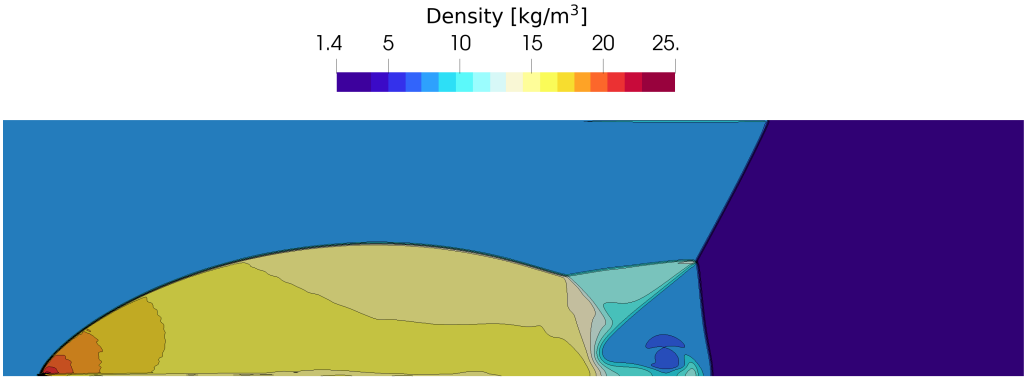


Figure 2: Density contours at t = 200 ms

12.2 Shock Tube - Two Fluid

The presented case is taken from Zheng et al. (2011) and consists of a gas using the van der Waals EOS and water using the Stiffened gas EOS initially separated at the center of the domain. The shock tube is 1 m long, and discretized into 300 computational cells.

The left state is defined as:

$$\rho = 1000 \text{ kg/m}^3 \quad \alpha = 1 \quad p = 1e9 \text{ Pa}.$$

The right state is defined as:

$$\rho = 50 \text{ kg/m}^3 \quad \alpha = 0 \quad p = 1e5 \text{ Pa}.$$

The `phaseProperties` dictionary is:

```
phases  (fluid gas);

fluid
{
    type          basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState stiffenedGas;
    }
    specie
    {
        molWeight   18.0;
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       4.4;
        a           6.0e8;
    }
    thermo
    {
        Cv          4186;
        Hf          0.0;
    }

    residualRho 1e-6;
    residualAlpha 1e-10;
}

gas
{
    type          basic;
```

```
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState vanderWaals;
    }
    specie
    {
        molWeight   28.97;
    }
    transport
    {
        mu          0;
        Pr          1.0;
    }
    equationOfState
    {
        gamma       1.4;
        a           5.0;
        b           1e-3;
    }
    thermo
    {
        Cv          718;
        Hf          0.0;
    }

    residualRho 1e-6;
    residualAlpha 1e-10;
}
```

and the `fvSchemes` dictionary is:

```
fluxScheme      HLLC;

ddtSchemes
{
    default         Euler;
```

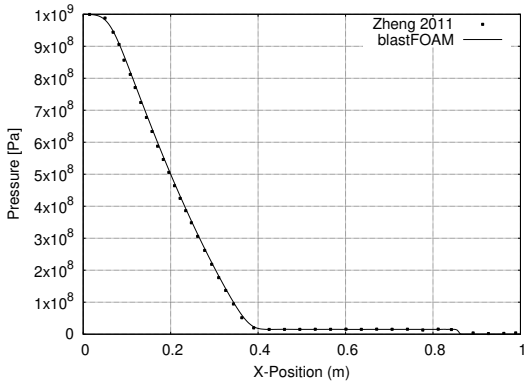
```
    timeIntegrator  RK2SSP;
}

gradSchemes
{
    default         leastSquares;
}

divSchemes
{
    default         none;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default             cubic;
    reconstruct(alpha)  vanLeer;
    reconstruct(rho)    vanLeer;
    reconstruct(U)      vanLeerV;
    reconstruct(e)      vanLeer;
    reconstruct(p)      vanLeer;
    reconstruct(speedOfSound) vanLeer;
}

snGradSchemes
{
    default         corrected;
}
```

As seen above, the HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 2nd-order strong-stability-preserving time integration. The following commands were used to initialize and run the case in serial:

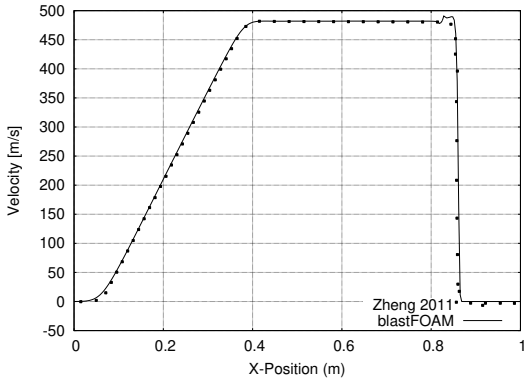
```
blockMesh
```

```
setFields
blastFoam
```

The results are shown in Fig. 3.



(a) Pressure



(b) X-velocity

Figure 3: Pressure and velocity fields at t = 73 ms

12.3 Two charge detonation

The next case is meant to show the capabilities of *blastFoam* to solve complex problems involving multiple phases and detonation dynamics. Two charges, one composed of RDX and the other of TNT are detonated in air. The RDX has a single detonation point at the center of its circular domain, and the TNT is detonated using the pressure based activation model. Fig. 4 shows

the initial charge where the red lines denote the outline of the RDX charge, the red dot the RDX initiation point, and the blue line the outline of the TNT charge.

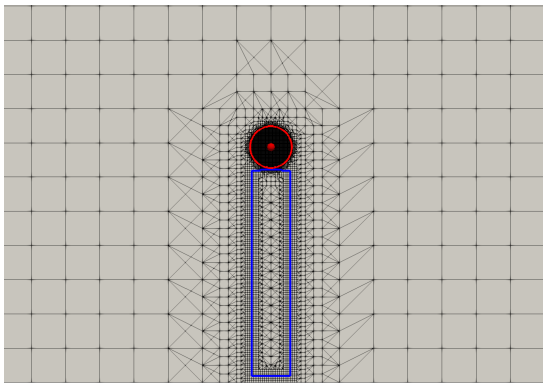


Figure 4: Initial charge configuration

The `phaseProperties` dictionary is:

```
phases (RDX tnt gas);

RDX
{
    type detonating;
    reactants
    {
        thermoType
        {
            transport   const;
            thermo      eConst;
            equationOfState Murnaghan;
        }
        equationOfState
        {
            rho0    1160;
            n       7.4;
            kappa   3.9e11;
            Gamma   0.35;
```

```
        pRef    101298;
    }
    specie
    {
        molWeight 222.12;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      1130.78;
        Hf      0.0;
    }
}
products
{
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState JWL;
    }
    equationOfState
    {
        A       2.9867e11;
        B       4.11706e9;
        R1      4.95;
        R2      1.15;
        C       7.206147e8;
        omega   0.35;
        rho0    1160;
    }
    specie
    {
        molWeight 55.0;
```

```
        }
        transport
        {
            mu      0;
            Pr      1;
        }
        thermodynamics
        {
            Cv      1000;
            Hf      0.0;
        }
    }

    activationModel none;
    initiation
    {
        E0      4.0e9;
    }

    residualRho     1e-6;
    residualAlpha   1e-10;
}

tnt
{
    type detonating;
    reactants
    {
        thermoType
        {
            transport   const;
            thermo      eConst;
            equationOfState BirchMurnaghan3;
        }
        equationOfState
        {
            rho0    1601; // Reference density [kg/m^3]
            K0      9.6e9; // Bulk modulus [Pa]
```

```
        K0Prime 6.6; // dK0/dp []
        Gamma   0.35; // Gruneisen coefficient
        pRef    101298; // Reference pressure [Pa]
    }
    specie
    {
        molWeight 227.13;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      1095;
        Hf      0.0;
    }
}
products
{
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState JWL;
    }
    equationOfState
    {
        rho0    1601;
        A       371.21e9;
        B       3.23e9;
        R1      4.15;
        R2      0.95;
        omega   0.3;
    }
    specie
    {
```

```
        molWeight 55.0;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      1000;
        Hf      0.0;
    }
}

activationModel pressureBased;
initiation
{
    E0      7.0e9; // Detonation energy [Pa]

    // Ignition coefficients
    I       0; // Ignition rate [1/s]

    // First stage coefficients
    G1      3.5083e-7; // Activation rate [Pa/s]
    c       1.0; // (1-lambda) exponent
    d       0.0; // lambda exponent
    y       1.3; // pressure exponent
    minLambda1  0.0; // 1st stage start value

    // Second stage coefficients
    G2      0.0; // Activation rate [Pa/s]

    pMin    1.1e5; // Cutoff pressure
}

residualRho     1e-6;
residualAlpha   1e-6;
}
```

```
gas
{
    type basic;
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState idealGas;
    }
    equationOfState
    {
        gamma   1.4;
    }
    specie
    {
        molWeight 28.97;
    }
    transport
    {
        mu      0;
        Pr      1;
    }
    thermodynamics
    {
        Cv      718;
        Hf      0;
    }

    residualRho     1e-6;
    residualAlpha   1e-6;
}
```

The HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 2nd-order strong-stability-preserving time integration. Below is the `fvSchemes` dictionary

```
fluxScheme      HLLC;
```

```
ddtSchemes
{
    default         Euler;
    timeIntegrator  RK2SSP;
}

gradSchemes
{
    default cellMDLimited leastSquares 1;
}

divSchemes
{
    default         none;
    div(alphaRhoPhi.tnt,lambda.tnt) Riemann;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default             cubic;
    reconstruct(alpha)  vanLeer;
    reconstruct(rho)    vanLeer;
    reconstruct(U)      vanLeerV;
    reconstruct(e)      vanLeer;
    reconstruct(p)      vanLeer;
    reconstruct(speedOfSound) vanLeer;
    reconstruct(lambda.tnt) vanLeer;
}

snGradSchemes
{
    default         corrected;
}
```

Because the circular charge domain is not well represented by the hexahedral mesh generated by *blockMesh*, refinement around this region is used. A maximum of four levels of refinement are used, with two cells between each level. This allows for a more accurate initial mass of the charges, and provides better initial resolution of the solution. The *setRefinedFields* utility is used where the `setFieldsDict` is

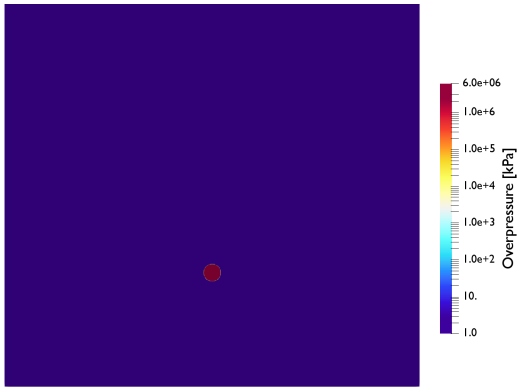
```
fields (alpha.RDX alpha.tnt);
nBufferLayers 1;


defaultFieldValues
(
    volScalarFieldValue alpha.gas       1
    volScalarFieldValue alpha.RDX       0
    volScalarFieldValue alpha.tnt       0
);

regions
(
    cylinderToCell
    {
        refineInternal yes;
        level 6;

        p1 (0 0.55 -1);
        p2 (0 0.55 1);
        radius 0.05;

        fieldValues
        (
            volScalarFieldValue alpha.RDX   1
            volScalarFieldValue alpha.gas   0
        );
    }
    boxToCell
    {
        level 5;
```

```
        refineInternal no;

        box (-0.05 0.01 -1) (0.05 0.5 1);

        fieldValues
        (
            volScalarFieldValue alpha.tnt   1
            volScalarFieldValue alpha.gas   0
        );
    }
);
```

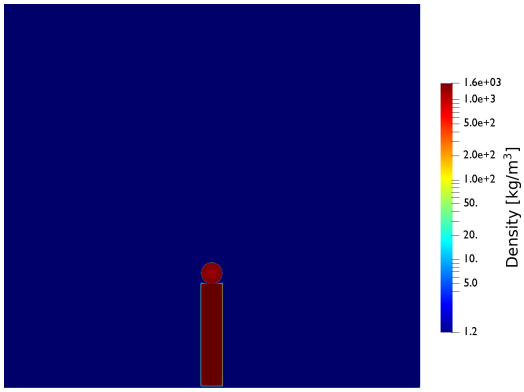
The following commands were used to run the case in parallel:

```
blockMesh
decomposePar
mpirun -np $nProcs setRefineFields -parallel
mpirun -np $nProcs blastFoam -parallel
```

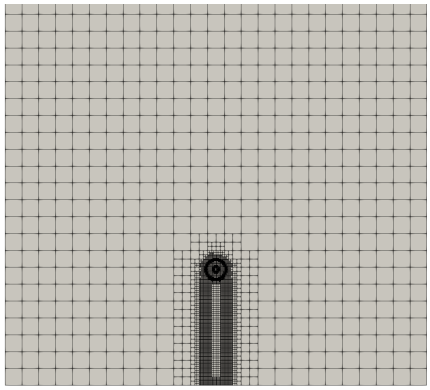
The instantaneous overpressure and density fields can be seen in Fig. 5, Fig. 6, and Fig. 7.



(a) Overpressure

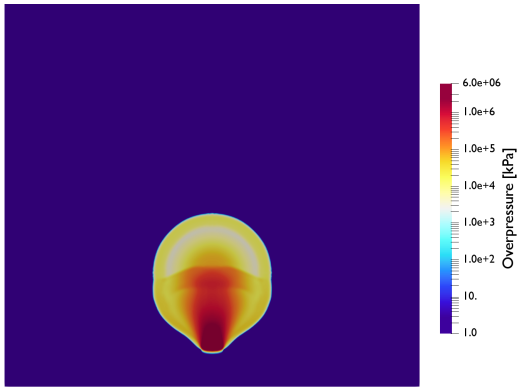


(b) Density

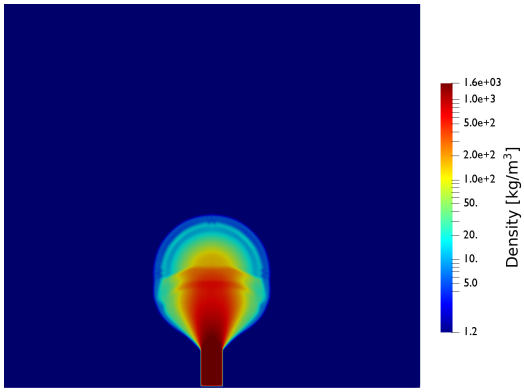


(c) Mesh

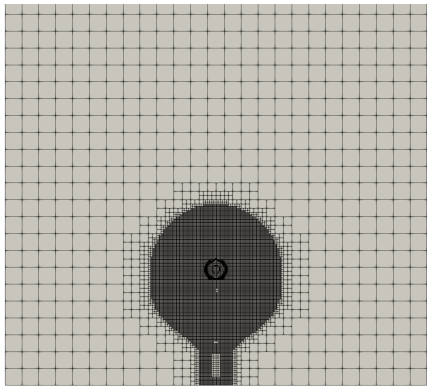
Figure 5: Instantaneous fields at t = 0.005 ms



(a) Overpressure

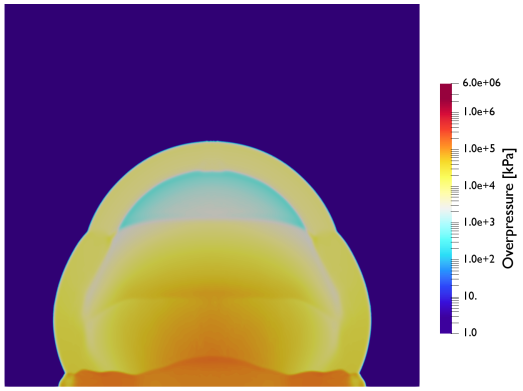


(b) Density

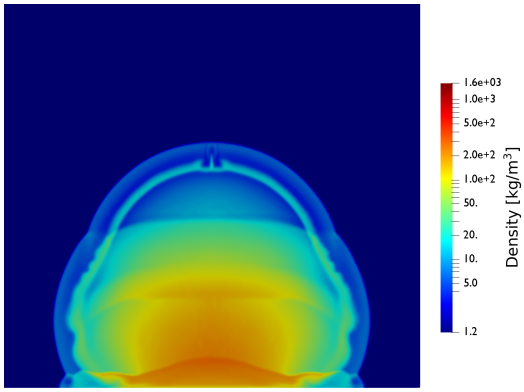


(c) Mesh

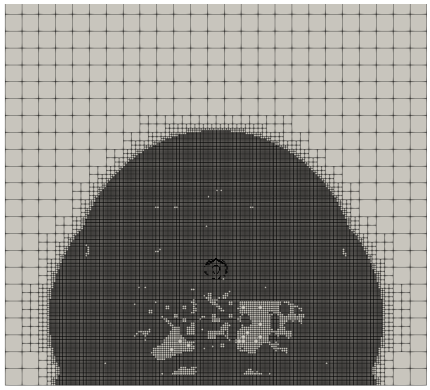
Figure 6: Instantaneous fields at t = 0.05 ms



(a) Overpressure



(b) Density



(c) Mesh

Figure 7: Instantaneous fields at t = 0.15 ms

12.4 Mine (buried charge)

The final case will illustrate how to use the *blastEulerFoam* solver to simulate an underground detonations. A small c4 charge is buried in a volume of particles (sand) and detonated, leading to the ejection of explosive gasses and particles in the the air above. The multiphase phase model will be used to represent the air and c4 gasses. The particle phase will be represented by the granular phase model with a density of 1470 kg/m^3 and a diameter of 50 μm. The `phaseProperties` file can be seen below

```
phases (particles gas);

particles
{
    phaseType        granular;

    diameterModel constant;
    d           50e-6;

    alphaMax    0.65;
    alphaMinFriction 0.5;

    type basic;
    thermoType
    {
        transport   constIso;
        thermo      eConst;
        equationOfState rhoConst;
    }
    specie
    {
        molWeight 100.0;
    }
    equationOfState
    {
        rho0     1470;
    }
    transport
```

```
    {
        kappa       237;
    }
    thermodynamics
    {
        Cv          987;            // Heat capacity
        Hf          0.0;
    }

    residualAlpha 1e-10;
    residualRho   1e-6;

    kineticTheoryCoeffs
    {
        e           0.9;
    }
}

gas
{
    phaseType       multiphase;
    diameterModel   constant;
    d               0;

    phases (c4 air);

    c4
    {
        type detonating;
        reactants
        {
            thermoType
            {
                transport   const;
                thermo      eConst;
                equationOfState idealGas;
            }
```

```
    equationOfState
    {
        rho0            1601;
        n               7.4;
        kappa           3.9e11;
        Gamma           0.35;
        gamma           1.35;
        pRef            101298;
    }
    specie
    {
        molWeight           55.0;
    }
    transport
    {
        mu              1.81e-5;              // Viscosity
        Pr              1;               // Prandtl number
    }
    thermodynamics
    {
        Cv          1400;           // Heat capacity
        Hf          0.0;
    }
}
products
{
    thermoType
    {
        transport   const;
        thermo      eConst;
        equationOfState JWL;
    }

    equationOfState
    {
        rho0      1601;
        A         609.77e9;
        B         12.95e9;
```

```
            R1          4.50;
            R2          1.4;
            omega       0.25;
        }
        specie
        {
            molWeight           55.0;
        }
        transport
        {
            mu              1.81e-5;              // Viscosity
            Pr              1;              // Prandtl number
        }
        thermodynamics
        {
            Cv          1400;           // Heat capacity
            Hf          0.0;
        }
    }
    residualRho     1e-6;           // Minimum density of the phase
    residualAlpha   1e-10;          // Minimum volume fraction used for divis

    activationModel linear;
    initiation
    {
        E0              9.0e9;
        points ((0 -1 0));
        vDet        7850;           // Detonation velocity [m/s]
    }
}

air
{
    type basic;
    thermoType
    {
        transport const;
        equationOfState idealGas;
```

```
            thermo eConst;
        }
        specie
        {
            molWeight 28.97;
        }
        equationOfState
        {
            gamma           1.4;
        }
        transport
        {
            mu              1.81e-5;              // Viscosity
            Pr              1;                // Prandtl number
        }
        thermodynamics
        {
            Cv          718;            // Heat capacity
            Hf          0.0;
        }
        residualRho     1e-6;
        residualAlpha   1e-10;
    }
}

blending
{
    default
    {
        type none;
        continuousPhase gas;
    }
}

interfacialPressure
(
    (gas and particles)
    {
```

```
        type single;
        phase gas;
    }
);

interfacialVelocity
(
    (gas and particles)
    {
        type single;
        phase particles;
    }
);

aspectRatio
(
);

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

virtualMass
(
);

heatTransfer
(
    (particles in gas)
```

```
    {
        type RanzMarshall;
    }
);

lift
(
);

wallLubrication
(
);

turbulentDispersion
(
);

kineticTheory
{
    residualAlpha       1e-6;

    radialModel SinclairJackson;
    viscosityModel Gidaspow;
    conductivityModel Gidaspow;
    granularPressureModel Lun;
    frictionalStressModel JohnsonJackson;

    JohnsonJacksonCoeffs
    {
        alphaMinFriction        0.5;
        Fr                      0.05;
        eta                     2;
        p                       5;
        phi                     28.5;
        alphaDeltaMin           1e-6;
    }
}
// ************************************************************************* //
```

and the `fvSchemes` dictionary is:

```
fluxSchemes
{
    gas
    {
        fluxScheme  HLLC;
    }
    particles
    {
        fluxScheme          AUSM+Up;
        alphaMinFriction    0.5;
    }
}

ddtSchemes
{
    default         Euler;
    timeIntegrator  RK2SSP 3;
}

gradSchemes
{
    default         cellMDLimited leastSquares 1;
}

divSchemes
{
    default         none;
    div(alphaRhoPhi.c4,lambda.c4) Gauss Riemann;
    div((((alpha.gas*rho.gas)*nuEff.gas)*dev2(T(grad(U.gas))))) Gauss linear;
    div(sigma.particles) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}
```

```
interpolationSchemes
{
    default             cubic;
    "reconstruct\(alpha.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(rho.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(U.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(e.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(p.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(speedOfSound.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(Theta.*\)"  quadraticMUSCL vanAlbada;
    "reconstruct\(lambda.*\)"  quadraticMUSCL vanAlbada;
}

snGradSchemes
{
    default         corrected;
}
```

Because the spherical charge is not well represented by the hexahedral mesh generated by *blockMesh*, refinement around this region is used. A maximum of four levels of refinement are used, with two cells between each level. This allows for a more accurate initial mass of the charges, and provides better initial resolution of the solution. The *setRefinedFields* utility is used, and the `setFieldsDict` is

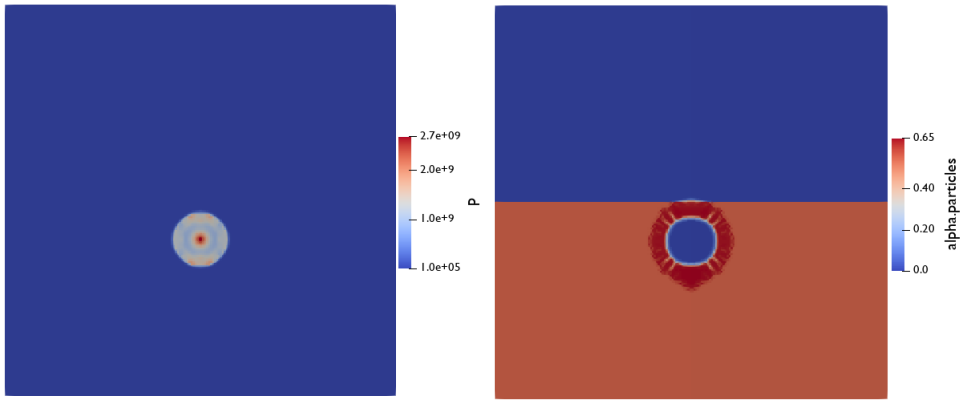
```
fields (alpha.particles);
nBufferLayers 1;

defaultFieldValues
(
    volScalarFieldValue alpha.particles 0
    volScalarFieldValue alpha.c4        0
    volScalarFieldValue alpha.air       1
);

regions
(
```

```
    boxToCell
    {
        level 1;
        box (-25 -25 -0.1) (25 0 0.1);

        fieldValues
        (
            volScalarFieldValue alpha.particles 0.55
        );
    }
    sphereToCell
    {
        centre (0 -1 0);
        radius 0.5;
        level 2;
        refineInternal yes;

        backup
        {
            centre (0 -1 0);
            radius 1;
        }

        fieldValues
        (
            volScalarFieldValue alpha.particles 0
            volScalarFieldValue alpha.c4        1
            volScalarFieldValue alpha.air       0
        );
    }
);
```

The following commands were used to run the case in parallel:

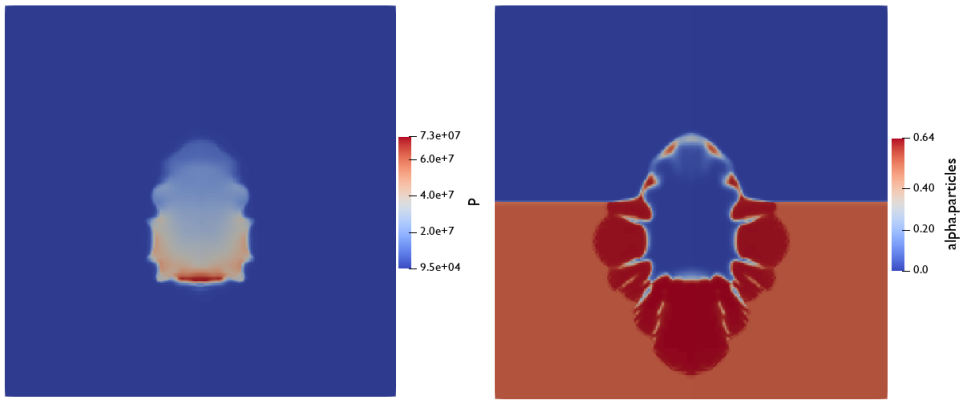
```
blockMesh
decomposePar
mpirun -np $nProcs setRefineFields -parallel
mpirun -np $nProcs blastEulerFoam -parallel
```

The instantaneous pressure and particle volume fraction fields can be seen in Fig. 8, Fig. 9, and Fig. 10. The case is simulated an axisymmetic domain, but the results have been mirrored.



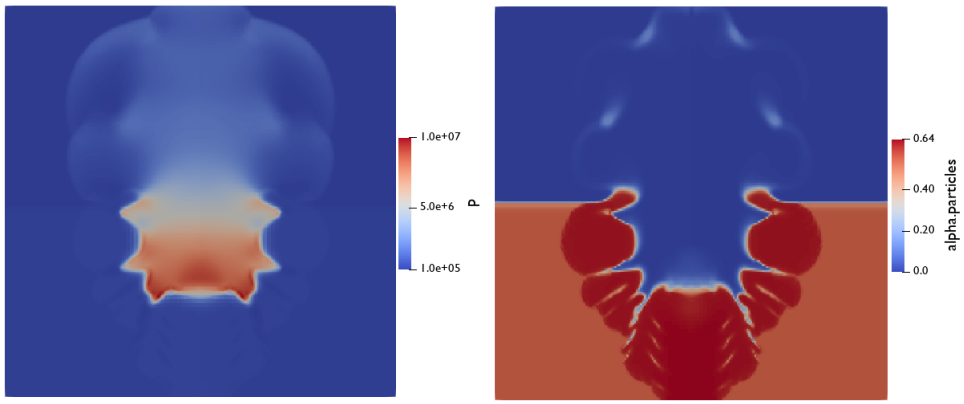
(a) Pressure (b) Particle volume fraction

Figure 8: Instantaneous fields at t = 0.0002 s



(a) Pressure (b) Particle volume fraction

Figure 9: Instantaneous fields at t = 0.002 s



(a) Pressure (b) Particle volume fraction

Figure 10: Instantaneous fields at t = 0.005 s

References

Agrawal, K., Loezos, P. N., Syamlal, M., and Sundaresan, S. (2001). The role of meso-scale structures in rapid gas–solid flows. *Journal of Fluid Mechanics*, 445:151 – 185.

Benedict, M., Webb, G. B., and Rubin, L. C. (1942). An Empirical Equation for Thermodynamic Properties of Light Hydrocarbons and Their Mixtures II. Mixtures of Methane, Ethane, Propane, and n-Butane. *The Journal of Chemical Physics*, 10(12):747–758. Publisher: American Institute of Physics.

Benyahia, S., Syamlal, M., and O'Brien, T. (2012). Summary of MFIX Equations 2012-1. Technical report.

Chao, Z., Wang, Y., Jakobsen, J. P., Fernandino, M., and Jakobsen, H. A. (2011). Derivation and validation of a binary multi-fluid Eulerian model for fluidized beds. *Chemical Engineering Science*, 66(16):3605–3616.

Chapman, S., Cowling, T. G., and Burnett, D. (1990). *The Mathematical Theory of Non-uniform Gases: An Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases*. Cambridge University Press. Google-Books-ID: Cbp5JP2OTrwC.

Doan, L. R. and Nickel, G. H. (1963). A subroutine for the equation of state of air. Technical Report RTD (WLR) TN63-2, Air Force Weapons Labratory.

Enwald, H., Peirano, E., and Almstedt, A.-E. (1996). Eulerian two-phase flow theory applied to fluidization. *International Journal of Multiphase Flow*, 22:21–66.

Fedors, R. F. and Landel, R. F. (1979). An Empirical method of estimating the void fraction in mixtures of uniform particles of different size. *Powder Technology*, 23(2):225–231.

Fox, R. O. (2019). A kinetic-based hyperbolic two-fluid model for binary hard-sphere mixtures. *Journal of Fluid Mechanics*, 877:282–329. Publisher: Cambridge University Press.

Gao, J., Chang, J., Lu, C., and Xu, C. (2008). Experimental and computational studies on flow behavior of gas–solid fluidized bed with disparately sized binary particles. *Particuology*, 6(2):59–71.

Gidaspow, D. (1994). *Multiphase Flow and Fluidization: Continuum and Kinetic Theory Descriptions*. Elsevier. Google-Books-ID: fHecceQyaYkC.

Giroux, E. D. (1973). *HEMP User's Manual*. Lawrence Livermore National Laboratory, University of California. Google-Books-ID: 3XgpHQAACAAJ.

Greenshields, C. J., Weller, H. G., Gasparini, L., and Reese, J. M. (2010). Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework, for high-speed viscous flows. *International journal for numerical methods in fluids*, 63(1):1–21.

Houim, R. W. and Oran, E. S. (2016). A multiphase model for compressible granular–gaseous flows: formulation and initial tests. *Journal of Fluid Mechanics*, 789:166–220.

Hrenya, C. M. and Sinclair, J. L. (1997). Effects of particle-phase turbulence in gas-solid flows. *AIChE Journal*, 43(4):853–869. _eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690430402.

Huilin, L. and Gidaspow, D. (2003). Hydrodynamics of binary fluidization in a riser: CFD simulation using two granular temperatures. *Chemical Engineering Science*, 58(16):3777–3792.

Johnston, I. A. (2005). The Noble-Abel Equation of State: Thermodynamic Derivations for Ballistics Modeling. Technical Report DSTO-TN-0670, Defense Science and Technology Organisation.

Kurganov, A. and Tadmor, E. (2000). New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations. *Journal of Computational Physics*, 160(1):241–282.

Lai, S., Houim, R. W., and Oran, E. S. (2018). Effects of particle size and density on dust dispersion behind a moving shock. *Physical Review Fluids*, 3(6):064306.

Liou, M.-S. (2006). A sequel to AUSM, Part II: AUSM+-up for all speeds. *Journal of Computational Physics*, 214(1):137–170.

Lun, C. K. K., Savage, S. B., Jeffrey, D. J., and Chepurniy, N. (1984). Kinetic theories for granular flow: inelastic particles in Couette flow and slightly inelastic particles in a general flowfield. *Journal of Fluid Mechanics*, 140:223–256.

Luo, H., Baum, J. D., and Löhner, R. (2004). On the computation of multimaterial flows using ALE formulation. *Journal of Computational Physics*, 194(1):304–328.

Lutzky, M. (1964). The flow field behind a spherical detonation in TNT using the landau-stanyukovich equation of state for detonation products. Technical report, NAVAL ORDNANCE LAB WHITE OAK MD.

Löhner, R. (1987). An adaptive finite element scheme for transient problems in CFD. *Computer Methods in Applied Mechanics and Engineering*, 61(3):323–338.

Madadi-Kandjani, E. and Passalacqua, A. (2015). An extended quadrature-based moment method with log-normal kernel density functions. *Chemical Engineering Science*, 131:323–339.

Mader, C. (1963). Detonation properties of condensed explosives computed using the becker-kistiakowsky-wilson equation of state.

Miller, P. J. (1995). A Reactive Flow Model with Coupled Reaction Kinetics for Detonation and Combustion in Non-Ideal Explosives. *MRS Online Proceedings Library Archive*, 418.

Needham, C. E. (2018). *Blast Waves*. Shock Wave and High Pressure Phenomena. Springer International Publishing, Cham.

OpenCFD Ltd. (2018a). *OpenFOAM - The Open Source CFD Toolbox - Programer's Guide*. United Kingdom, 2 edition.

OpenCFD Ltd. (2018b). *OpenFOAM - The Open Source CFD Toolbox - User's Guide*. United Kingdom, 2 edition.

Passalacqua, A., Laurent, F., Madadi-Kandjani, E., Heylmun, J. C., and Fox, R. O. (2018). An open-source quadrature-based population balance solver for OpenFOAM. *Chemical Engineering Science*, 176:306–318.

Rettenmaier, D., Deising, D., Ouedraogo, Y., Gjonaj, E., De Gersem, H., Bothe, D., Tropea, C., and Marschall, H. (2019). Load balanced 2D and 3D adaptive mesh refinement in OpenFOAM. *SoftwareX*, 10:100317.

Schiller, L. and Naumann, A. (1933). Über die grundlegenden Berechnungen bei der Schwerkraftaufbereitung. *Ver. Deut. Ing.*, 77:318–320.

Shyue, K.-M. (2001). A Fluid-Mixture Type Algorithm for Compressible Multicomponent Flow with Mie–Grüneisen Equation of State. *Journal of Computational Physics*, 171(2):678–707.

Sinclair, J. L. and Jackson, R. (1989). Gas-particle flow in a vertical pipe with particle-particle interactions. *AIChE Journal*, 35(9):1473–1486.

Souers, P. C., Anderson, S., Mercer, J., McGuire, E., and Vitello, P. (2000). JWL++: A Simple Reactive Flow Code Package for Detonation. page 5.

Spiteri, R. J. and Ruuth, S. J. (2002). A New Class of Optimal High-Order Strong-Stability-Preserving Time Discretization Methods. *SIAM Journal on Numerical Analysis*, 40(2):469–491.

Syamlal, M., Rogers, W., and O'Brien, T. (1993). MFIX documentation theory guide. Technical Report DOE/METC–94/1004, 10145548.

Toro, E. F., Spruce, M., and Speares, W. (1994). Restoration of the contact surface in the HLL-Riemann solver. *Shock waves*, 4(1):25–34.

Wardlaw, A., Mckeown, R., and Luton, A. (1998). Coupled hydrocode prediction of underwater explosion damage.

Woodward, P. and Colella, P. (1984). The numerical simulation of two-dimensional fluid flow with strong shocks. *Journal of Computational Physics*, 54(1):115–173.

Xie, W., Zhang, R., Lai, J., and Li, H. (2019). An accurate and robust HLLC-type Riemann solver for the compressible Euler system at various Mach numbers. *International Journal for Numerical Methods in Fluids*, 89(10):430–463.

Yu, A. B. and Standish, N. (1987). Porosity calculations of multi-component mixtures of spherical particles. *Powder Technology*, 52(3):233–241.

Yuan, C. and Fox, R. O. (2011). Conditional quadrature method of moments for kinetic equations. *Journal of Computational Physics*, 230(22):8216–8246.

Zheng, H. W., Shu, C., Chew, Y. T., and Qin, N. (2011). A solution adaptive simulation of compressible multi-fluid flows with general equation of state. *International Journal for Numerical Methods in Fluids*, 67(5):616–637.

Zheng, Z. and Zhao, J. (2016). Unreacted equation of states of typical energetic materials under static compression: A review. *Chinese Physics B*, 25(7):076202.