blastFoam Theory and User Guide

Version 5.0

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

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

July 10, 2022

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, including single- and multi-velocity systems, 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. Many of the available models include options that the user can tune to best suit their simulations, and are given in a table at the end of each model description.

1.1 System Requirements

`blastFoam` currently builds against OpenFOAM-9[2]. The following dependencies are required to successfully compile and run `blastFoam`:

1. A working installation of OpenFOAM-9 complete with libraries and headers produced by OpenFOAM-9, no other packages are required to run `blastFoam` applications. If OpenFOAM-9 is compiled from source, the *ThirdParty-9* repository source is also required. If a pre-compiled version of OpenFOAM-9 is used, this is automatically installed.

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. **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.

[2] https://github.com/OpenFOAM/OpenFOAM-9

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 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:

```
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.

Next run the **./Allwmake** command from the top `blastfoam` directory to compile the libraries and applications. Finally, ensure that OpenFOAM-9 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 if if does not exist
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

# Source the etc/bashrc to your .bashrc file
# by runngin the following command in terminal
echo "source $HOME/OpenFOAM/blastfoam/etc/bashrc" >> $HOME/.bashrc

# Re-load and set the bash environment to compile blastFoam
source $HOME/.bashrc

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

1.4 Executable

The applications to solve these equations are executed by running the executable name (i.e. **blastFoam**). The executables are stored within the *$BLAST_APPBIN* (*$FOAM_USER_APPBIN* by default but this can be overridden) directory and can be run from any directory.

1.5 Getting Help

This guide is intended to cover the major points of the library. For more specific questions regarding the equations or models, the references listed in the class header files (*.H) should be consulted.

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: (1) the single-fluid case where a single shared velocity and energy is used and (2) 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 whether the single-fluid or multi-fluid is used. Each of these are presented in the following sections.

2.1 Single-fluid model

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

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho Y_s \\ \rho \mathbf{u} \\ \rho E \end{pmatrix} \tag{2.2}$$

$$\mathbf{F} = \begin{pmatrix} \rho \mathbf{u} \\ \rho Y_s \mathbf{u} \\ \rho \mathbf{u} \otimes \mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix} \tag{2.3}$$

$$\mathbf{S} = \begin{pmatrix} 0 \\ \dot{R}_s + \boldsymbol{\nabla} \cdot \mathbf{J}_s \\ \dot{\mathrm{M}} + \dot{\mathrm{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix} \tag{2.4}$$

where Y_s is the mass fraction of specie s, and $\mathbf{J}_s$ is the diffusion flux of species s. For multiple phases

$$\mathbf{U} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ \alpha_1\rho_1 \\ \vdots \\ \alpha_n\rho_n \\ \rho\mathbf{u} \\ \rho E \end{pmatrix} \tag{2.5}$$

$$\mathbf{F} = \begin{pmatrix} \alpha_1\mathbf{u} \\ \vdots \\ \alpha_n\mathbf{u} \\ \alpha_1\rho_1\mathbf{u} \\ \vdots \\ \alpha_n\rho_n\mathbf{u} \\ \rho\mathbf{u}\otimes\mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix} \tag{2.6}$$

$$\mathbf{S} = \begin{pmatrix} \alpha_1\boldsymbol{\nabla}\cdot\mathbf{u} \\ \vdots \\ \alpha_n\boldsymbol{\nabla}\cdot\mathbf{u} \\ 0 \\ \vdots \\ 0 \\ \dot{\mathbf{M}} + \dot{\mathbf{M}}_v \\ \dot{\mathrm{E}} + \dot{\mathrm{E}}_v \end{pmatrix} \tag{2.7}$$

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

$$\sum_i \alpha_i = 1 \tag{2.8}$$

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

$$\alpha_2 = 1 - \alpha_1 \tag{2.9}$$

Additionally, the mixture density is defined by

$$\rho = \sum_i \alpha_i \rho_i \tag{2.10}$$

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

$$\dot{\mathbf{M}}_v = \boldsymbol{\nabla} \cdot \left(\rho\nu \left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^\intercal - \frac{2}{3}(\boldsymbol{\nabla} \cdot \mathbf{u})\mathbf{I} \right] \right) \tag{2.11}$$

and

$$\dot{\mathrm{E}}_v = \boldsymbol{\nabla} \cdot \left(\rho\nu \left[\boldsymbol{\nabla}\mathbf{u} + (\boldsymbol{\nabla}\mathbf{u})^\intercal - \frac{2}{3}(\boldsymbol{\nabla} \cdot \mathbf{u})\mathbf{I} \right] \cdot \mathbf{u} \right) + \boldsymbol{\nabla} \cdot (\alpha_T \boldsymbol{\nabla} e) \tag{2.12}$$

where 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, a shared internal energy and temperature is used, and individual phase masses are conserved. 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. In contrast, however, 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 i^{th} fluid phase are

$$\mathbf{U}_i = \begin{pmatrix} \alpha_i \\ \alpha_i \rho_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i E_i \end{pmatrix} \tag{2.13}$$

$$\mathbf{F} = \begin{pmatrix} \alpha_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \otimes \mathbf{u}_i + \alpha_i p_i \mathbf{I} \\ \alpha_i (\rho_i E_i + p_i) \mathbf{u}_i \end{pmatrix} \tag{2.14}$$

$$\mathbf{S} = \begin{pmatrix} \alpha_i \boldsymbol{\nabla} \cdot \mathbf{u}_i + \sum_{i \neq j} \frac{\dot{\omega}_{ij}}{I(\rho_i, \rho_j)} \\ \sum_{i \neq j} \dot{\omega}_{ij} \\ \alpha_i \rho_i \mathbf{g} + \sum_{i \neq j} \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \alpha_i \rho_i \mathbf{g} \cdot \mathbf{u}_i + \sum_{i \neq j} (\dot{\mathrm{E}}_{ij} + \dot{\mathbf{M}}_{ij} \cdot \mathbf{u}_{\text{int},ij}) + \dot{\mathrm{E}}_{\text{det},i} + \dot{\mathrm{E}}_{v,i} \end{pmatrix} \tag{2.15}$$

where α_i is the phase volume fraction, ρ_i is the phase density, $\mathbf{u}_i$ is the phase velocity, E_i is the total phase energy, and p_i is the phase pressure. The source terms, $\dot{\mathbf{M}}_{ij}$ and $\dot{\mathrm{E}}_{ij}$, are the momentum and energy sources that account for interactions between other phases, $\dot{\mathbf{M}}_{v,i}$ and $\dot{\mathrm{E}}_{v,i}$ are viscous terms, $\mathbf{g}$ is the gravitational acceleration vector, $\dot{\mathrm{E}}_{\text{det},i}$ is the energy added due to reactions, and $\mathbf{u}_{\text{int},ij}$ is the interfacial velocity between two phases. When mass transfer is used $\dot{\omega}_{i,j}$ is the mass transfer rate from the i^{th} phase to the j^{th} phase, and I is a switching function used to select the transferring quantity (Xue et al., 2011) and is defined as

$$I(x_i, x_j) = \xi_{i,j} x_j + (1 - \xi_{i,j}) x_i \tag{2.16}$$

and

$$\xi_{i,j} = \begin{cases} 1 & \text{if } \dot{\omega}_{i,j} > 0 \\ 0 & \text{else} \end{cases} \tag{2.17}$$

The viscous stress uses a Newtonian stress tensor defined as

$$\dot{\mathbf{M}}_{v,i} = \boldsymbol{\nabla} \cdot \left(\alpha_i \rho_i \nu_i \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^{\intercal} - \frac{2}{3} (\boldsymbol{\nabla} \cdot \mathbf{u}_i) \mathbf{I} \right] \right) \tag{2.18}$$

and

$$\dot{\mathrm{E}}_{v,i} = \boldsymbol{\nabla} \cdot \left(\alpha_i \rho_i \nu_i \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^{\intercal} - \frac{2}{3} (\boldsymbol{\nabla} \cdot \mathbf{u}_i) \mathbf{I} \right] \cdot \mathbf{u}_i \right) + \boldsymbol{\nabla} \cdot (\alpha_i \alpha_{T,i} \boldsymbol{\nabla} e_i) \tag{2.19}$$

where 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

$$\dot{\mathbf{M}}_{ij} = \dot{\mathbf{M}}_{\mathrm{D},ij} + \dot{\mathbf{M}}_{\mathrm{L},ij} + \dot{\mathbf{M}}_{\mathrm{VM},ij} + \dot{\mathbf{M}}_{\mathrm{WL},ij} + \dot{\mathbf{M}}_{\mathrm{TD},ij} + \dot{\omega}_{i,j} I(\mathbf{u}_i, \mathbf{u}_j) \tag{2.20}$$

where the summed terms, in order, represent drag, lift, virtual mass, wall lubrication, and turbulent dispersion, respectively.
The energy transfer term, $\dot{\mathrm{E}}_{ij}$, is defined as

$$\dot{\mathrm{E}}_{ij} = \dot{\mathrm{E}}_{\mathrm{HT},ij} + \dot{\omega}_{i,j} I(E_i, E_j + h_{f_i} - h_{f_j}) \tag{2.21}$$

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

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

$$\mathbf{U}_i = \begin{pmatrix} \alpha_i \rho_i \\ \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i e_i \\ \alpha_i \rho_i \langle E \rangle_i \end{pmatrix} \tag{2.22}$$

$$\mathbf{F} = \begin{pmatrix} \alpha_i \rho_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \otimes \mathbf{u}_i + (P_{s,i} + P_{\text{fric},i})\mathbf{I} \\ \alpha_i \rho_i e_i \mathbf{u}_i \\ \alpha_i \rho_i \mathbf{u}_i \langle E \rangle_i \end{pmatrix} \tag{2.23}$$

$$\mathbf{S} = \begin{pmatrix} \sum_{i \neq j} \dot{\omega}_{ij} \\ \alpha_i \rho_i \mathbf{g} - \alpha_i \sum_\ell \boldsymbol{\nabla} p_{\text{int},i\ell} + \sum_j \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \sum_j \dot{\mathrm{e}}_{ij} + \sum_k \dot{\gamma}_{ik} + \dot{e}_{\text{det},i} \\ -\sum_k \dot{\gamma}_{ik} - \sum_\ell \phi_{\text{prod},i\ell} \\ -P_{s,i} \boldsymbol{\nabla} \cdot \mathbf{u}_i + \alpha_i \boldsymbol{\tau}_i : \boldsymbol{\nabla} \mathbf{u}_i \\ + \boldsymbol{\nabla} \cdot (\kappa_{s,i} \boldsymbol{\nabla} \Theta_i) + \sum_{i \neq j} \dot{\omega}_{ij} I(\langle E \rangle_i, 0) \end{pmatrix} \tag{2.24}$$

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 $\sum_\ell$ denote summations over all phases, all solid phases, and all fluid phases, respectively. The new quantities, $P_{s,i}$, $P_{\text{fric},i}$, $\langle E \rangle$, 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, $\dot{\mathbf{M}}_{ij}$, $\dot{\mathrm{e}}_{ij}$, $\dot{\gamma}_{ij}$, and $\phi_{\text{prod},il}$ are the momentum transfer, energy transfer, 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 $\dot{\mathrm{e}}_{\text{det},i} = \dot{\mathrm{E}}_{\text{det},i}$). 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 become

$$\mathbf{S} = \begin{pmatrix} \alpha_i \boldsymbol{\nabla} \cdot \mathbf{u}_i + \sum_{i \neq j} \frac{\dot{\omega}_{ij}}{I(\rho_i, \rho_j)} \\ \sum_{i \neq j} \dot{\omega}_{ij} \\ \alpha_i \rho_i \mathbf{g} + \sum_{i \neq j} \dot{\mathbf{M}}_{ij} + \dot{\mathbf{M}}_{v,i} \\ \alpha_i \rho_i \mathbf{g} \cdot \mathbf{u}_i + \sum_{i \neq j} \dot{\mathrm{E}}_{ij} + \dot{\mathrm{E}}_{\mathrm{det},i} + \dot{\mathrm{E}}_{v,i} + \sum_k \phi_{\mathrm{prod},k} \\ \sum_k \dot{\omega}_{ik} I(0, \langle E \rangle_k) \end{pmatrix} \tag{2.25}$$

You can also see in Eq. (2.22) 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 be generated due to the loss of energy from inelastic collisions.

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

$$\dot{\mathbf{M}}_{ij} = \boldsymbol{\nabla} \cdot \boldsymbol{\tau}_{s,i} + \dot{\omega}_{i,j} I(\mathbf{u}_i, \mathbf{u}_j) \tag{2.26}$$

and the granular stress is defined as

$$\boldsymbol{\tau}_{s,i} = \rho_i \nu_{s,i} \left[\boldsymbol{\nabla} \mathbf{u}_i + (\boldsymbol{\nabla} \mathbf{u}_i)^{\intercal} \right] + \left(\lambda_{s,i} - \frac{2}{3} \rho_i \nu_{s,i} \right) (\boldsymbol{\nabla} \cdot \mathbf{u}_i) \mathbf{I} \tag{2.27}$$

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

$$\lambda_{s,i} = \sum_k P_{s,ik} \frac{d_{ik}}{3} \sqrt{\frac{2(m_i \Theta_i + m_k \Theta_k)^2}{\pi \Theta_i \Theta_k (m_i^2 \Theta_i + m_k^2 \Theta_k)}} \tag{2.28}$$

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

$$P_{s,i} = \alpha_i \rho_i \Theta_i + \sum_k P_{s,ik} \tag{2.29}$$

The production of granular energy is given by

$$\phi_{\mathrm{prod},i\ell} = \frac{81 \alpha_i \mu_\ell^2}{g_{0,ii} d_i^3 \rho_i \sqrt{\pi}} \frac{|\mathbf{u}_\ell - \mathbf{u}_i|^2}{\sqrt{\Theta_i}} - 3 K_{D,i\ell} \Theta_i \tag{2.30}$$

where $K_{D,i\ell}$ 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)

$$\dot{\gamma}_{ik} = \left[\frac{3}{d_{ik}}\left(\frac{2m_0^2\Theta_i\Theta_k}{\pi(m_i^2\Theta_i + m_k^2\Theta_k)}\right)^{1/2} - \frac{3m_0(m_i\Theta_i + m_k\Theta_k)}{4(m_i^2\Theta_i + m_k^2\Theta_k)}\boldsymbol{\nabla}\cdot\mathbf{u}_i\right](1-e_{ik})P_{s,ik} \tag{2.31}$$

where

$$m_i = \frac{\rho_i\pi d_i^3}{6} \tag{2.32}$$

$d_{ik} = (d_i + d_k)/2$, $m_0 = m_i + m_k$, 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 k. The drag, radial distribution function, and granular pressure are determined by models and will be discussed in Sec. 3.1 and Sec. 4. The specific manner as to how these source terms are solved can be found in Appendix B.7 of Houim and Oran (2016).
The energy transfer term, $\dot{\mathrm{e}}_{ij}$, is defined as

$$\dot{\mathrm{e}}_{ij} = \dot{\mathrm{e}}_{\mathrm{HT},ij} + \dot{\omega}_{i,j}I(e_i, e_j + h_{f_i} - h_{f_j}) \tag{2.33}$$

Lastly, the speed of sound for a granular material is given by

$$c_i^2 = \frac{1}{\rho_i}\left[\frac{\partial P_{\mathrm{tot},i}}{\partial\alpha_i} + \frac{2}{3}\frac{\Theta_i\left(\frac{\partial P_{\mathrm{tot},i}}{\partial\Theta_i}\right)^2}{\rho_i\alpha_i^2}\right] \tag{2.34}$$

where $P_{\mathrm{tot},i} = P_{s,i} + P_{\mathrm{fr},i}$. 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 is not discussed here.

The topic of granular flows, or kinetic theory, is quite extensive and thus it would be impractical to attempt to provide complete descriptions and derivations of the equations and models discussed here. For more more details on this topic, the reader should consult the following resources:

- For derivations relating to molecular gasses, see Chapman et al. (1990)
- For more practical derivations, see Gidaspow (1994)
- For applications to compressible flows, see Fox (2019)
- For derivations of the polydisperse formulations, see Huilin and Gidaspow (2003) and Chao et al. (2011)

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. Drag and heat transfer, however, are solved using an analytical solution; the reference used for these solution methods will be provided in the relevant sections.

NOTE: There are more models currently implemented that do not apply to gas-particle flows and, therefore, are not presented at this time. Their descriptions and equations will be added as they become relevant.

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, $K_{\mathrm{D},ij}$:

$$K_{\mathrm{D},ij} = 0.75 \frac{\alpha_d C_{\mathrm{D}} \mathrm{Re}_{ij} \rho_c \nu_c}{d_d^2} \tag{3.1}$$

where the Reynolds number between phases i and j is defined as $\mathrm{Re}_{ij} = \frac{d_d |\mathbf{u}_c - \mathbf{u}_d|}{\nu_c}$. The subscripts c and d are used to denote the continuous and dispersed phases, respectively. The term $C_D \mathrm{Re}_{ij}$ will be defined by the models discussed in the following sections.

The momentum transfer term is defined as $\dot{\mathbf{M}}_{\mathrm{D},ij} = K_{\mathrm{D},ij}(\mathbf{u}_j - \mathbf{u}_i)$. 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, though this approach is much more computationally expensive.

Variable	Description
swarmCorrelation	Swarm model

3.1.1 SchillerNaumann

The drag model of Schiller and Naumann (1933) calculates the drag coefficient for flow over a single sphere:

$$C_{\mathrm{D}} \mathrm{Re}_{ij} = \begin{cases} 24(1 + 0.15 \mathrm{Re}_{ij}^{0.687}) & \text{if } \mathrm{Re}_{ij} < 1000 \\ 0.44 \mathrm{Re}_{ij} & \text{else} \end{cases} \tag{3.2}$$

Variable	Description
residualRe	Minimum Reynolds number [ ]

3.1.2 WenYu

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

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = C_{\mathrm{D}} Res_{SN,ij} \alpha_c^{-2.65} \tag{3.3}$$

where $C_{\mathrm{D}}\mathrm{Re}_{SN,ij}$ is the Schiller Naumann 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.,

$$Res_{ij} = \mathrm{Re}_{ij}\alpha_c \tag{3.4}$$

Variable	Description
residualRe	Minimum Reynolds number [ ]

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

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = \frac{4}{3}\frac{150(1-\alpha_c)}{\alpha_c} + 1.75\mathrm{Re}_{ij} \tag{3.5}$$

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

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = \frac{4}{3}\left(\frac{17.3}{\alpha_2} + 0.336\mathrm{Re}_{ij}\right)\alpha_c\alpha_2^{-2.8} \tag{3.6}$$

where $\alpha_2 = 1 - \alpha_d$.

3.1.5 SyamlalOBrien

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

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = \frac{C_{\mathrm{D}s}\mathrm{Re}_{ij}\alpha_c}{V_r^2} \tag{3.7}$$

where

$$C_{\mathrm{D}s}\mathrm{Re}_{ij} = \left(0.63\sqrt{\mathrm{Re}_{ij}} + 4.8\sqrt{V_r}\right)^2 \tag{3.8}$$

$$V_r = 0.5\left(A - 0.06\mathrm{Re}_{ij} + \sqrt{(0.06\mathrm{Re}_{ij})^2 + 0.12\mathrm{Re}_{ij}(2B - A) + A^2}\right) \tag{3.9}$$

$$A = \alpha_2^{4.14} \tag{3.10}$$

$$B = \begin{cases} 0.8\alpha_2^{1.28} & \text{if } \alpha_2 < 0.85 \\ \alpha_2^{2.65} & \text{else} \end{cases} \tag{3.11}$$

and $\alpha_2 = 1 - \alpha_d$.

3.1.6 GidaspowErgunWenYu

The drag model of 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):

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = \begin{cases} C_{\mathrm{D}}\mathrm{Re}_{\mathrm{WenYu},ij} & \text{if } \alpha_c > 0.80 \\ C_{\mathrm{D}}\mathrm{Re}_{\mathrm{Ergun},ij} & \text{else} \end{cases} \tag{3.12}$$

where $C_{\mathrm{D}}\mathrm{Re}_{\mathrm{WenYu},ij}$ is the Wen Yu drag coefficient and $C_{\mathrm{D}}\mathrm{Re}_{\mathrm{Ergun},ij}$ is the Ergun drag coefficient, both multiplied by the Reynolds number.

Variable	Description
residualRe	Minimum Reynolds number [ ]

3.1.7 GidaspowSchillerNaumann

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

$$C_{\mathrm{D}}\mathrm{Re}_{ij} = C_{\mathrm{D}s}\mathrm{Re}_{ij}\alpha_c^{-1.65} \tag{3.13}$$

$$C_{\mathrm{D}s}\mathrm{Re}_{ij} = \begin{cases} \frac{24(1+0.15Res_{ij}^{0.687})}{\alpha_c} & \text{if } Res_{ij} < 1000 \\ 0.44Res_{ij} & \text{else} \end{cases} \tag{3.14}$$

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

$$Res_{ij} = \mathrm{Re}_{ij}\alpha_c \tag{3.15}$$

Variable	Description
residualRe	Minimum Reynolds number [ ]

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

$$\dot{\mathbf{M}}_{\mathrm{L},ij} = C_{\mathrm{L},ij}\alpha_d\rho_c\left[(\mathbf{u}_c - \mathbf{u}_i) \times (\boldsymbol{\nabla} \times \mathbf{u}_c)\right] \tag{3.16}$$

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

3.2.1 constant

The lift coefficient is a constant value.

Variable	Description
Cl	Lift coefficient [ ]

3.3 Heat transfer models

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

$$\mathrm{Pr}_{ij} = \frac{\nu_c C_{vc} \rho_c}{\kappa_c} \tag{3.17}$$

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).

3.3.1 RanzMarshall

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

$$\dot{\mathrm{E}}_{\mathrm{HT},ij} = \frac{6\alpha_d \kappa_c \mathrm{Nu}_{ij}}{d_d^2} \tag{3.18}$$

where the Nusselt number is a run-time selectable model, and all other properties are taken from the thermodynamic model.

Variable	Description
NuModel	Nusselt number model
residualAlpha	Stabilization value of volume fraction (defaults to that of the phaseModel)

3.3.2 noneNu

This should only be used is a Nusselt number is needed by some other model, but heat transfer is neglected.

3.4 Nusselt number models

3.4.1 constant

For the constant Nusselt number heat transfer model, Eq. (3.18) is again used, but the Nusselt number is a constant value provided by the user.

Variable	Description
Nu	Nusselt number [ ]

3.4.2 duct

For hollow particles the Nusselt number of Dittus and Boelter (1930) is defined as

$$\mathrm{Nu}_{ij} = 0.023\mathrm{Re}_{ij}^{0.8}\mathrm{Pr}_{ij}^{0.4} \tag{3.19}$$

3.4.3 GelperinEinstein

The Nusselt number of Gelperin and Einstein (1971) is defined as

$$\mathrm{Nu}_{ij} = 0.023\mathrm{Re}_{ij}^{2/3}\mathrm{Pr}_{ij}^{1/3} \tag{3.20}$$

3.4.4 Gunn

The Nusselt number of Gunn (1978) is defined as

$$\begin{aligned}\mathrm{Nu}_{ij} =&(7 - 10\alpha_c + 5.0\alpha_c^2)(1 + 0.7\mathrm{Re}_{ij}^{0.2}\mathrm{Pr}_{ij}^{1/3}) \\ &+(1.33 - 2.4\alpha_c + 1.2\alpha_c^2)\mathrm{Re}_{ij}^{0.7}\mathrm{Pr}_{ij}^{1/3}\end{aligned} \tag{3.21}$$

3.4.5 RanzMarschall

The Ranz-Marschall Nusselt number for spheres is defined as

$$\mathrm{Nu}_{ij} = 2 + 0.6\mathrm{Re}_{ij}^{1/2}\mathrm{Pr}_{ij}^{1/3} \tag{3.22}$$

3.4.6 sphere

The Nusselt number for spheres is defined as

$$\mathrm{Nu}_{ij} = 2 + 0.4\mathrm{Re}_{ij}^{1/2}\mathrm{Pr}_{ij}^{1/3} \tag{3.23}$$

3.5 Mass transfer models

Currently on particle to fluid mass transfer has been implemented. Fluid-fluid mass transfer is not currently supported.

3.5.1 none

No mass transfer.

3.5.2 reactingParticle

The mass transfer rate is explicitly calculated using the change in mass of a particle where the regression rate is determined by a reaction rate. the *reactingParticle* diameter model must be used.

3.6 Interfacial pressure models

The interfacial pressure model is used to determine the pressure at the interface between two phases.

3.6.1 single

The pressure of the given phase is used.

Variable	Description
phase	Name of the phase

NOTE: For gas-particle flows this option should be used, using the gas phase pressure.

3.6.2 volumeAveraged

The volume fraction averaged pressure of the two phases is used.

3.6.3 totalPressure

The volume fraction averaged thermodynamic pressure and the mass averaged dynamic pressure is used.

3.7 Interfacial velocity models

The interfacial velocity model is used to determine the velocity at the interface between two phases.

3.7.1 single

The velocity of the given phase is used.

Variable	Description
phase	Name of the phase

NOTE: For gas-particle flows this option should be used, using the particle phase velocity.

3.7.2 massAveraged

The mass averaged velocity of the two phases is used.

4 Granular 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.

Some common variables used in the granular models will now be defined. Here, $\sum_k$ 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

$$\mathbb{S} = \frac{1}{2}\left[(\boldsymbol{\nabla}\mathbf{u})^\intercal + \boldsymbol{\nabla}\mathbf{u}\right] - \frac{1}{3}(\boldsymbol{\nabla}\cdot\mathbf{u})\mathbb{I} \tag{4.1}$$

The symmetric strain is defined as

$$\mathbb{D} = \frac{1}{2}\left[(\boldsymbol{\nabla}\mathbf{u})^\intercal + \boldsymbol{\nabla}\mathbf{u}\right] \tag{4.2}$$

Finally, the total granular drag on a phase is defined as

$$K_{\mathrm{D},i} = \sum_k K_{\mathrm{D},ik} \tag{4.3}$$

4.1 Radial distribution models

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

4.1.1 CarnahanStarling

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

$$g_{0,ii} = \frac{1}{1-\alpha_i} + \frac{3\alpha_i}{2(1-\alpha_i)^2} + \frac{\alpha_i^2}{2(1-\alpha_i)^3} \tag{4.4}$$

4.1.2 SinclairJackson

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

$$g_{0,ii} = \frac{1}{1 - \left(\frac{\alpha_i}{\alpha_{i,\max}}\right)^{1/3}} \tag{4.5}$$

4.1.3 LunSavage

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

$$g_{0,ii} = \left(1 - \frac{\alpha_i}{\alpha_{\max,i}}\right)^{-2.5\alpha_{\max,i}} \tag{4.6}$$

4.1.4 Gao

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

$$g_{0,ij} = \frac{d_i g_{0,i} + d_j g_{0,j}}{d_i + d_j} \tag{4.7}$$

$$g_{0,i} = \frac{1}{1 - \left(\frac{\alpha_i}{\alpha_{i,\max}}\right)^{1/3}} \tag{4.8}$$

4.1.5 Lebowitz

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

$$g_{0,ij} = \frac{1}{1 - \alpha_s} + \frac{3 d_i d_j}{(1 - \alpha_s)^2 (d_i + d_j)} \sum_k \frac{\alpha_k}{d_k} \tag{4.9}$$

4.2 Granular viscosity models

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

4.2.1 Gidaspow

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

$$\begin{aligned}\nu_{s,i} = &\frac{4}{5}\alpha_i^2 g_{0,ii}\frac{(1+e_{i,j})}{\pi} + \frac{1}{15}\sqrt{\pi}g_{0,ii}(1+e_{ii})\alpha_i^2 \\ &+ \frac{1}{6}\sqrt{\pi}\alpha_i + \frac{10}{96}\frac{\sqrt{\pi}}{(1+e_{ii})g_{0,ii}}\end{aligned} \tag{4.10}$$

4.2.2 HrenyaSinclair

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

$$\begin{aligned}\nu_{s,i} = &\frac{4}{5}\alpha_i^2 g_{0,ii}\frac{(1+e_{i,j})}{\pi} \\ &+ \frac{1}{15}\sqrt{\pi}g_{0,ii}(3e_{ii}-1)\frac{\alpha_i^2}{3-e_{ii}} \\ &+ \frac{1}{6}\frac{\sqrt{\pi}\alpha_i(0.5\lambda + 0.25(3e_{ii}-1))}{0.5(3-e_{ii})\lambda} \\ &+ \frac{10}{96}\frac{\sqrt{\pi}}{(1+e_{ii})0.5(3-e_{ii})g_{0,ii}\lambda}\end{aligned} \tag{4.11}$$

where

$$\lambda = 1 + \frac{d_i L}{6\sqrt{2}\alpha_i} \tag{4.12}$$

Variable	Description
L	characteristic length [m]

4.2.3 Princeton

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

$$\nu_{s,i} = \rho_i \frac{3.6}{3}\left[\frac{\mu_i^*}{g_{0,ii}\eta(2-\eta)}\left(1+\frac{8}{5}\eta\sum_k(\alpha_k g_{0,ik})\right)\right.$$
$$\left.\times\left(1+\frac{8}{5}\eta(3\eta-2)\sum_k(\alpha_k g_{0i,k})\right)+\frac{3}{5}\eta\mu_b\right] \quad (4.13)$$

where

$$\mu_i^* = \frac{\rho_i\alpha_i g_{0,ii}\Theta_i\mu}{\rho_i\sum_k(\alpha_k g_{0,ik})\Theta_i + \frac{2K_{d,i\mu}}{\rho_i\alpha_i}} \quad (4.14)$$

$$\mu = \frac{5}{96}\rho_i d_i\sqrt{\pi\Theta_i} \quad (4.15)$$

and

$$\mu_b = \frac{256}{5\pi}\mu\alpha_i\sum_k(\alpha_k g_{0,ik}) \quad (4.16)$$

4.2.4 Syamlal

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

$$\nu_{s,i} = \frac{4}{5}\alpha_i^2 g_{0,ii}\frac{(1+e_{ii})}{\pi} + \frac{1}{15}\sqrt{\pi}g_{0,ii}(3e_{ii}-1)\frac{\alpha_i^2}{3-e_{ii}} + \frac{1}{6}\frac{\sqrt{\pi}\alpha_i}{(3-e_{ii})} \quad (4.17)$$

4.2.5 none

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

4.3 Granular conductivity models

4.3.1 Chao

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

$$\kappa_{s,i} = \frac{2\kappa_{\text{dilute},i}}{\frac{1}{N}\sum_k (1+e_{ik})g_{0,ik}} \left[1 + \frac{6}{5}\sum_k \alpha_k (1+e_{ik}) g_{0,ik}\right]^2 \tag{4.18}$$

where N is the number of granular phases and

$$\kappa_{\text{dilute},i} = \frac{15\rho_i \Theta_i}{\sum_k \left[\pi\sqrt{2\pi} n_k d_{ij}^2 (\sqrt{\Theta_i} + \sqrt{\Theta_j} - 0.56\sqrt[4]{\Theta_i \Theta_j})\right]} \tag{4.19}$$

4.3.2 Gidaspow

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

$$\begin{aligned}\kappa_{s,i} = {} & 2\alpha_i^2 g_{0,ii} \frac{(1+e_{ii})}{\sqrt{\pi}} + \frac{9}{8}\sqrt{\pi} g_{0,ii} 0.5(1+e_{ii})\alpha_i^2 \\ & + \frac{15}{16}\sqrt{\pi}\alpha_i + \frac{25}{64}\frac{\sqrt{\pi}}{g_{0,ii}(1+e_{ii})}\end{aligned} \tag{4.20}$$

4.3.3 HrenyaSinclair

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

$$\begin{aligned}\kappa_{s,i} = {} & 2\alpha_i^2 g_{0,ii} \frac{(1+e_{ii})}{\sqrt{\pi}} \\ & + \frac{9}{8}\frac{\sqrt{\pi} g_{0,ii} 0.25(1+e_{ii})^2 (2e_{ii}-1)\alpha_i^2}{\frac{49}{16} - \frac{33}{16}e_{ii}} \\ & + \frac{15}{16}\frac{\sqrt{\pi}\alpha_i (0.5e_{ii}^2 + 0.25e_{ii} - 0.75 + \lambda)}{\left(\frac{49}{16} - \frac{33}{16}e_{ii}\right)\lambda} \\ & + \frac{25}{64}\frac{\sqrt{\pi}}{g_{0,ii}\lambda(1+e_{ii})\left(\frac{49}{16} - \frac{33}{16}e_{ii}\right)}\end{aligned} \tag{4.21}$$

where

$$\lambda = 1 + \frac{d_i L}{6\sqrt{2}\alpha_i} \tag{4.22}$$

Variable	Description
L	characteristic length [m]

4.3.4 Princeton

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

$$\kappa_{s,i} = \frac{\kappa_i^*}{g_{0,ii}} \left[\left(1 + \frac{12}{5}\eta \sum_k (\alpha_k g_{0,ik}) \right) \left(1 + \frac{12}{5}\eta^2 (4\eta - 3) \sum_k (\alpha_k g_{0,ik}) \right) + \frac{64}{25\pi}(41 - 33\eta)\eta^2 \left(\sum_k (\alpha_k g_{0,ik}) \right)^2 \right] \tag{4.23}$$

where

$$\kappa_i^* = \frac{\rho_i \alpha_i g_{0,ii} \Theta_i \kappa_i}{\rho_i \sum_k (\alpha_k g_{0,ik}) \Theta_i + \frac{6 K_{d,i} \kappa_i}{5 \rho_i \alpha_i}} \tag{4.24}$$

and

$$\kappa_i = \frac{75 \rho_i d_i \sqrt{\pi \Theta_i}}{48 \eta (41 - 33\eta)} \tag{4.25}$$

4.3.5 Syamlal

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

$$\kappa_{s,i} = 2\alpha_i^2 g_{0,ii} \frac{(1 + e_{ii})}{\sqrt{\pi}} + \frac{9}{8} \frac{\sqrt{\pi} g_{0,ii} 0.25 (1 + e_{ii})(2e_{ii} - 1)\alpha_i^2}{\frac{49}{16} - 3316 e_{ii}} + \frac{15}{16} \frac{\sqrt{\pi}\alpha_i}{\frac{49}{16} - \frac{33}{16} e_{ii}} \tag{4.26}$$

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.

4.4.1 Huilin

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

$$\begin{aligned} P_{s,ij} = & \frac{\pi(1+e_{ij})d_{ij}^3 g_{0,ij} n_i n_j m_i m_j m_0 \Theta_i \Theta_j}{3(m_i^2\Theta_i + m_j^2\Theta_j)} \\ & \times \left[\frac{m_0^2 \Theta_i \Theta_j}{(m_i^2\Theta_i + m_j^2\Theta_j)(\Theta_i + \Theta_j)}\right]^{3/2} \\ & \times (1 - 3\omega + 6\omega^2 - 10\omega^3) \end{aligned} \tag{4.27}$$

with

$$\omega = \frac{m_i\Theta_i - m_j\Theta_j)}{[(m_i^2\Theta_i^2 + m_j^2\Theta_j^2) + \Theta_i\Theta_j(m_i^2 + m_j^2)]^{1/2}} \tag{4.28}$$

4.4.2 Lun

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

$$P_{s,ij} = \alpha_i \alpha_j \rho_i \Theta_i 2(1+e_{ij}) g_{0,ij} \tag{4.29}$$

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.

4.5.1 JohnsonJackson

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

$$P_{\text{fr}} = \begin{cases} Fr\frac{(\alpha_s - \alpha_{\text{crit},s})^{\eta}}{(\alpha_{max,s} - \alpha_s)^{p}} & \text{if } \alpha_s > \alpha_{\text{crit},s} \\ 0 & \text{else} \end{cases} \tag{4.30}$$

and the frictional viscosity is

$$\nu_{\text{fr}} = 0.5 P_{\text{fr}} \sin(\phi) \tag{4.31}$$

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. $\alpha_{\text{crit},s}$ is the granular particle fraction that the frictional pressure "turns on", and $\alpha_{\text{max},s}$ is the maximum packing limit of all granular phases. Generally $\alpha_{\text{crit},s}$ is taken to be around 0.5.

Variable	Description
Fr	Model coefficient [ ]
eta	Model coefficient [ ]
p	Model coefficient [ ]
phi	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{max,s}$ [ ]
alphaMinFriction	Volume fraction friction is “turned on” [ ]

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, $\alpha_{\text{crit},s}$ 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 ratio of the local volume fraction by the critical volume fraction is used, i.e. $\alpha_{\text{crit},s} = \left(\frac{\alpha_{\text{crit},s}}{\alpha_{\text{max},s}}\right)_{\text{crit}} \alpha_{\text{max},s}$ where $\left(\frac{\alpha_{\text{crit},s}}{\alpha_{\text{max},s}}\right)_{\text{crit}}$ is specified by the user and $\alpha_{\text{max},s}$ is the calculated local packing limit. The pressure is defined as

$$P_{\text{fr}} = \begin{cases} 10^{24}(\alpha_s - \alpha_{\text{crit},s})^{10} & \text{if } \alpha_s > \alpha_{\text{crit},s} \\ 0 & \text{else} \end{cases} \tag{4.32}$$

and the viscosity is

$$\nu_{\text{fr}} = \begin{cases} 0.5\frac{P_{\text{fr}}\sin(\phi)}{\sqrt{\frac{1}{3}(I_1(\mathbb{D})^2 - I_2(\mathbb{D}))}} & \text{if } \alpha_s > \alpha_{\text{crit},s} \\ 0 & \text{else} \end{cases} \tag{4.33}$$

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

Variable	Description
phi	Model coefficient [ ]
alphaMinFrictionByAlphap	Volume fraction ratio friction is "turned on" [ ]

4.5.3 JohnsonJacksonSchaeffer

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.

$$P_{\text{fr}} = \begin{cases} Fr\frac{(\alpha_s - \alpha_{\text{crit},s})^\eta}{(\alpha_{\text{max},s} - \alpha_s)^p} & \text{if } \alpha_s > \alpha_{\text{crit},s} \\ 0 & \text{else} \end{cases} \tag{4.34}$$

$$\nu_{\text{fr}} = \begin{cases} 0.5\frac{P_{\text{fr}}\sin(\phi)}{\sqrt{\frac{1}{3}(I_1(\mathbb{D})^2 - I_2(\mathbb{D}))}} & \text{if } \alpha_s > \alpha_{\text{crit},s} \\ 0 & \text{else} \end{cases} \tag{4.35}$$

Variable	Description
Fr	Model coefficient [ ]
eta	Model coefficient [ ]
p	Model coefficient [ ]
phi	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{\text{max},s}$ [ ]
alphaMinFriction	Volume fraction friction is "turned on" [ ]

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 $\alpha_{\text{crit,JJ},s}$ but less

than $\alpha_{\text{crit,Sc},s}$. As with the Schaeffer model, the critical ratio of $\left(\frac{\alpha_{\text{crit,Sc},s}}{\alpha_{\max,s}}\right)_{\text{crit}}$ is used as an input rather than $\alpha_{\text{crit,Sc},s}$.

$$P_{\text{fr}} = \begin{cases} 10^{24}(\alpha_s - \alpha_{\text{crit,Sc},s})^{10} & \text{if } \alpha_s > \alpha_{\text{crit,Sc},s} \\ Fr\frac{(\alpha_s - \alpha_{\text{crit,JJ},s})^{\eta}}{(\alpha_{\max,s} - \alpha_s)^{p}} & \text{if } \alpha_{\text{crit,Sc},s} \leq \alpha_s > \alpha_{\text{crit,JJ},s} \\ 0 & \text{else} \end{cases} \tag{4.36}$$

The frictional viscosity is defined using

$$\frac{P_{c,k}}{P_{\text{fr}}} = \left(1 - \frac{\boldsymbol{\nabla} \cdot \mathbf{u}_s}{n\sqrt{2}\sin(\phi)\sqrt{\mathbb{S} : \mathbb{S} + \Theta_k/d_k^2}}\right)^{n-1} \tag{4.37}$$

$$\nu_{\text{fr},k} = \frac{\sqrt{2}P_{c,k}\sin(\phi)}{\sqrt{\mathbb{S} : \mathbb{S} + \Theta_k/d_k^2}}\left[n - (n-1)\left(\frac{P_{c,k}}{P_{\text{fr}}}\right)^{\frac{1}{1-n}}\right]\frac{\alpha_k}{\alpha_s} \tag{4.38}$$

and

$$n = \begin{cases} \frac{\sqrt{3}}{2\sin(\phi)} & \text{if } \boldsymbol{\nabla} \cdot \mathbf{u}_s \geq 0 \\ 1.03 & \text{else} \end{cases} \tag{4.39}$$

Variable	Description
Fr	Model coefficient [ ]
eta	Model coefficient [ ]
p	Model coefficient [ ]
phi	Model coefficient [ ]
alphaDeltaMin	Minimum value of $\alpha_s - \alpha_{\max,s}$ [ ]
alphaMinFriction	Volume fraction friction is "turned on" [ ]
alphaMinFrictionByAlphap	Volume fraction ratio friction is "turned on" [ ]

4.6 Packing limit models

The modeling of the packing limit is important in polydisperse flows due to the fact that, when large variations 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.

4.6.1 constant

A constant value for the packing limit is used.

Variable	Description
alphaMax	Packing limit [ ]

4.6.2 YuStandish

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

$$\alpha_{\max,s} = \min_{i \in 1:N} \left[\frac{\alpha_{\max,i}}{1 - \sum_{k=1}^{i-1} \left(1 - \frac{\alpha_{\max,i}}{p_{ik}}\right) \frac{cx_i}{X_{ik}} - \sum_{k=i+1}^{N} \left(1 - \frac{\alpha_{\max,i}}{p_{ik}}\right) \frac{cx_i}{X_{ik}}} \right] \tag{4.40}$$

where

$$cx_i = \frac{\alpha_i}{\alpha_s} \tag{4.41}$$

$$X_{ik} = \begin{cases} \frac{1-r_{ik}^2}{2-\alpha_{\max,i}} & \text{if } k < i \\ 1 - \frac{1-r_{ik}^2}{2-\alpha_{\max,i}} & \text{if } k \geq i \end{cases} \tag{4.42}$$

$$p_{ik} = \begin{cases} \alpha_{\max,i} + \alpha_{\max,i}(1 - \alpha_{\max,i}) & \\ \quad \times (1 - 2.35 r_{ik} + 1.35 r_{ik}^2) & \text{if } r_{ik} \leq 0.741 \\ \alpha_{\max,i} & \text{if } r_{ik} > 0.741 \end{cases} \tag{4.43}$$

and

$$r_{ik} = \begin{cases} \frac{d_i}{d_k} & \text{if } i \geq k \\ \frac{d_k}{d_i} & \text{if } i < k \end{cases} \tag{4.44}$$

Variable	Description
residualAlpha	Minimum volume fraction to divide by [ ]

4.6.3 FedorsLandel

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

$$\alpha_{\max,s} = \begin{cases} [(\alpha_{\max,1} - \alpha_{\max,2}) & \\ \quad +(1 - \sqrt{r_{21}}(1 - \alpha_{\max,1})\alpha_{\max,2}] & \\ \quad \times[\alpha_{\max,1} + (1 - \alpha_{\max,1})\alpha_{\max,2}]\frac{cx_i}{\alpha_{\max,1}} & \\ \quad +\alpha_{\max,2} & \text{if } cx_1 \leq \frac{\alpha_{\max,1}}{\alpha_{\max,1}+(1-\alpha_{\max,1})\alpha_{\max,2}} \\ (1 - \sqrt{r_{21}})[\alpha_{\max,1} & \\ \quad +(1 - \alpha_{\max,1})\alpha_{\max,2}]cx_2 + \alpha_{\max,1} & \text{else} \end{cases} \tag{4.45}$$

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

Variable	Description
residualAlpha	Minimum volume fraction to divide by [ ]

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.

4.7.1 Chao

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

$$\begin{aligned} K_{D,ij} = \frac{\alpha_i\alpha_j\rho_i\rho_j}{m_0} & d_{ij}(1 + e_{ij})g_{0,ij} \\ & \times \left[\sqrt{2\pi}(\sqrt{\Theta_i} + \sqrt{\Theta_j}) - \sqrt{2}(\Theta_i\Theta_j)^{1/4} + 0.5\pi|\mathbf{u}_i - \mathbf{u}_j|\right. \\ & \left. - 1.135\sqrt{|\mathbf{u}_i - \mathbf{u}_j|}\left(\sqrt{4}\Theta_i + \sqrt[4]{\Theta_j} - 0.8\sqrt[8]{\Theta_i\Theta_j}\right)\right] \end{aligned} \tag{4.46}$$

4.7.2 Syamlal

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

$$K_{D,ij} = \frac{3(1+e_{ij})(\frac{\pi}{2} + \frac{C_{f,ij}\pi^2}{8})\alpha_i\alpha_j\rho_i\rho_j(d_i+d_j)^2 g_{0,ij}|\mathbf{u}_i - \mathbf{u}_j|}{2\pi(\rho_i d_i^2 + \rho_j d_j^3)} + C_1 P_{fr} \tag{4.47}$$

where $C_{f,ij}$ is the coefficient of friction between granular phases i and j, and C_1 is the coefficient of frictional pressure.

Variable	Description
C1	Coefficient of frictional pressure (default is 0) [s/m^2]

5 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, is the solid thermodynamic model, which is only used for solid phases within multi-fluid solvers and multi-region solvers. These two models will share the basic thermodynamic classes (eConst, hConst, etc), as well as the choice in the type (basic, detonating, or multicomponent), however the solid phase only can use a constant density equation of state, and the transport model describes the 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      eConst;
        equationOfState idealGas;
    }
    specie
    {
        molWeight   28.97;
    }
    transport
    {
        ...
    }
```

```
    thermodynamics
    {
        ...
    }
    equationOfState
    {
        gamma       1.4;
        ...
    }
}
```

NOTE: Not all combinations of thermodynamic model, equation of state, transport, and mixture type are compiled, however using the dynamic compilation available in OpenFOAM, if a valid option is requested it will be compiled at run time and the library will be stored in the dynamicCode folder in the case. If any changes are made to the source code will not affect these libraries, and the dynamicCode folder will need to be removed for re-compilation.

5.1 Thermodynamic models

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

$$e_T = \int Cv(T)\mathrm{dT} \tag{5.1}$$

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

$$e_{\mathrm{eos}}(\rho, T) = \int \left(\frac{\partial p(\rho, T)}{\partial T} \right) \mathrm{d}v \tag{5.2}$$

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

$$e = e_T(T) + e_{\mathrm{eos}}(\rho, T) \tag{5.3}$$

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. All transport models, with the exception of the tabulated methods were ported from OpenCFD Ltd. (2018b).

$$C_v = C_p - CpMCv \tag{5.4}$$

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

$$CpMCv = \frac{\partial(p(\rho, e)v)}{\partial T} \tag{5.5}$$

For an ideal gas, $CpMCv = \bar{R}/W$ where $\bar{R}$ is the ideal gas constant (8314 [J/mol/K]) and W 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.
When enthalpy is used, the internal energy is calculated using

$$e = h - \frac{p}{\rho} \tag{5.6}$$

NOTE: By default the internal energy is limited so that the temperature is greater than 0. If *limitT* is set to *no* the temperature and energy will not be limited. Caution should be used with this option.

5.1.1 eConst

Internal energy is calculated using a constant value of specific heat at constant volume.

$$e_T(T) = C_v(T - T_{ref}) + E_{s,ref} \tag{5.7}$$

Variable	Description
Cv	Specific at constant volume [J/kg/K]
Tref	Reference temperature [K] (293.15 K default)
Esref	Reference sensible internal energy [J/kg/] (T_{ref} C_v default)
Hf	Heat of formation [J/kg]

5.1.2 ePolynomial

Internal energy is calculated using polynomial with powers of T.

$$e_T(T) = \sum_{i=0}^{7} C_{v,i} T^i \tag{5.8}$$

Variable	Description
CvCoeffs<8>	Specific heat at constant volume polynomial coefficients
Hf	Heat of formation [J/kg]
Sf	Standard entropy [J/kg/K]

5.1.3 ePower

Internal energy using a power function of temperature.

$$e_T(T) = C_v(T)T \tag{5.9}$$

$$C_v(T) = C_0 \left(\frac{T}{T_{ref}} \right)^{n_0} \tag{5.10}$$

Variable	Description
C0	Reference heat capacity at constant volume [J/kg/K]
n0	Exponent of the power function
Tref	Reference temperature [K]
Hf	Heat of formation [J/kg]

5.1.4 eIcoTabulated

The incompressible tabulated thermodynamic model can be used to calculate the temperature using temperature to lookup the internal energy. The data is read from a one dimensional table. The specific heat are calculated using finite difference derivatives to the tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x" and "f" names are "T" and "e" respectively. A heat of formation, "Hf", must also be provided.

5.1.5 eTabulated

The tabulated thermodynamic model can be used to calculate the temperature using density and temperature to find internal energy. The data is read from a two dimensional table. The specific heat are calculated using finite difference derivatives to the tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x", "y", and "f" names are "rho", "T", and "e" respectively. A heat of formation, "Hf", must also be provided.
NOTE: The equation of state contribution of internal energy is not included so this must be included in the tables provided.

5.1.6 hConst

Enthalpy is calculated using a constant value of specific heat at constant pressure.

$$h_T(T) = C_p(T - T_{ref}) + H_{s,ref} \tag{5.11}$$

Variable	Description
Cp	Specific at constant pressure [J/kg/K]
Tref	Reference temperature [K] (293.15 K default)
Hsref	Reference sensible enthalpy [J/kg/] (T_{ref} C_p default)
Hf	Heat of formation [J/kg]

5.1.7 hPolynomial

Enthalpy is calculated using polynomial with powers of T.

$$h_T(T) = \sum_{i=0}^{7} C_{p,i} T^i \tag{5.12}$$

Variable	Description
CpCoeffs<8>	Specific heat at constant pressure polynomial coefficients
Hf	Heat of formation [J/kg]
Sf	Standard entropy [J/kg/K]

5.1.8 hPower

Internal energy using a power function of temperature.

$$h_T(T) = C_p(T)T \tag{5.13}$$

$$C_v(T) = C_0 \left(\frac{T}{T_{ref}} \right)^{n_0} \tag{5.14}$$

Variable	Description
C0	Reference heat capacity at constant volume [J/kg/K]
n0	Exponent of the power function
Tref	Reference temperature [K]
Hf	Heat of formation [J/kg]

5.1.9 hIcoTabulated

The incompressible tabulated thermodynamic model can be used to calculate the temperature using temperature to lookup the enthalpy. The data is read from a one dimensional table. The specific heat are calculated using finite difference derivatives to the tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x" and "f" names are "T" and "h" respectively. A heat of formation, "Hf", must also be provided.

5.1.10 hTabulated

The tabulated thermodynamic model can be used to calculate the temperature using density and temperature to find enthalpy. The data is read from a two dimensional table. The specific heat are calculated using finite difference derivatives to the tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x", "y", and "f" names are "rho", "T", and "h" respectively. A heat of formation, "Hf", must also be provided.
NOTE: The equation of state contribution of internal energy is not included so this must be included in the tables provided.

5.1.11 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]

5.2 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.

5.2.1 rhoConst

For a solid phase, it is assumed to have a constant density, ρ_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] (default 101235 Pa)

5.2.2 Internal energy based (Mie-Grüneisen form)

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

$$p = (\Gamma_i - 1)\rho_i e - \Pi_i \tag{5.15}$$

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

$$c_i^2 = \frac{\Gamma_i p + \Pi_i}{\rho_i} - \delta_i(\Gamma_i - 1) \tag{5.16}$$

where δ_i is defined

$$\delta_i = -\frac{[p + \Pi_i]\Gamma_i'}{[\Gamma_i - 1]^2} + \frac{\Pi_i'}{\Gamma_i - 1} \tag{5.17}$$

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

5.2.2.1 idealGas

For an ideal gas, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i \tag{5.18}$$

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

$$\Gamma_i = \gamma_i \tag{5.19}$$

$$\Pi_i = 0 \tag{5.20}$$

$$\delta_i = 0 \tag{5.21}$$

Variable	Description
gamma	Specific heat ratio [ ]

5.2.2.2 stiffenedGas

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

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i a_i \tag{5.22}$$

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

$$\Gamma_i = \gamma_i \tag{5.23}$$

$$\Pi_i = \gamma_i a_i \tag{5.24}$$

and

$$\delta_i = 0 \tag{5.25}$$

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

5.2.2.3 (linearTillotson

The linear Tillotson equation of state was presented by (Wardlaw et al., 1998), and the pressure is defined as

$$p_i = p_{0,i} + \omega_i \rho_i (e_T - e_{0,i}) + A_i \mu + B_i \mu^2 + C_i \mu^3 \tag{5.26}$$

where A_i, B_i, C_i, and ω_i are material coefficients, and $p_{0,i}$ and $e_{0,i}$ are the reference pressure and internal energy. μ is defined as $\rho_i/\rho_{0,i} - 1$. The cavitation pressure, p_cav is used as the minimum pressure which occurs when the density is sufficiently below the reference density.

In the Mie-Grüneisen form

$$\Gamma_i = \omega_i + 1 \tag{5.27}$$

and

$$\Pi_i = \omega_i \rho_i e_{0,i} - p_{0,i} - A_i \mu - B_i \mu^2 - C_i \mu^3 \tag{5.28}$$

and

$$\delta_i = e_{0,i} - \frac{A + 2B\mu + 3.0C\mu^2}{\rho_{0,i}\omega_i} \tag{5.29}$$

Variable	Description
p0	Reference pressure [Pa]
rho0	Reference density [kg/m^3]
e0	Reference internal energy [kJ/kg]
omega	Model coefficient [ ]
A	Model coefficient [Pa]
B	Model coefficient [Pa]
C	Model coefficient [Pa]
pCav	Cavitation pressure [Pa]

5.2.2.4 Tillotson

The Tillotson equation of state was originally developed for metals by Tillotson (1962), and the pressure is defined as

$$p_i = \max\left(\rho_i e\left(a + \frac{b}{\frac{e}{e_{0,i}\eta^2} + 1}\right) + A_i\mu + B_i\mu^2, p_{cav}\right) \tag{5.30}$$

where A_i, B_i, C_i, and ω_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 $\rho_i/\rho_{0,i} - 1$. The cavitation pressure, p_cav is used as the minimum pressure which occurs when the density is sufficiently below the reference density.

In the Mie-Grüneisen form

$$\Gamma_i = \frac{a_i x + b_i e_{0,i}\rho_i^2}{x} - \frac{bee_{0,i}(\rho_i\rho_{0,i})^2}{x^2} + 1 \tag{5.31}$$

where

$$x_i = e\rho_{0,i}^2 + e_{0,i}\rho_i^2 \tag{5.32}$$

and

$$\Pi_i = -A_i\mu - B_i\mu^2 - b_i\frac{(e\rho_i\rho_{0,i})^2 e_{0,i}\rho_i}{\left(e\rho_{0,i}^2 + e_{0,i}\rho_i^2\right)^2} \tag{5.33}$$

and δ_i is calculated using (5.17)

Variable	Description
rho0	Reference density [kg/m^3]
e0	Reference internal energy [kJ/kg]
a	Model coefficient [ ]
b	Model coefficient [ ]
A	Model coefficient [Pa]
B	Model coefficient [Pa]
pCav	Cavitation pressure [Pa]

5.2.2.5 Tait

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

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i(b_i - a_i) \tag{5.34}$$

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

$$\Gamma_i = \gamma_i \tag{5.35}$$

$$\Pi_i = \gamma_i(b_i - a_i) \tag{5.36}$$

and

$$\delta_i = 0 \tag{5.37}$$

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

5.2.2.6 vanderWaals

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

$$p_i = \frac{\gamma_i - 1}{1 - b_i\rho_i}(\rho_i e_i + a_i\rho_i^2) - (a_i\rho_i^2 + c_i) \tag{5.38}$$

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

$$\Gamma_i = \frac{\gamma_i - 1}{1 - b_i \rho_i} + 1 \tag{5.39}$$

$$\Pi_i = \left[1 - \frac{\gamma_i - 1}{1 - b_i \rho_i}\right] a_i \rho_i^2 + \left[\frac{\gamma_i - 1}{1 - b_i \rho_i} + 1\right] c_i \tag{5.40}$$

and

$$\delta_i = -b_i \frac{p_i + a_i \rho_i^2}{\gamma_i - 1} + \left(\frac{1 - b_i \rho_i}{\gamma - 1} - 1\right) 2 a_i \rho_i \tag{5.41}$$

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

5.2.2.7 LSZK

The Landau, Stanyukovich, Zeldovich, and Kompaneets 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

$$\Gamma_i = \gamma_i \tag{5.42}$$

$$\Pi_i = a_i \rho_i^{b_i} \tag{5.43}$$

and

$$\delta_i = 0 \tag{5.44}$$

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

5.2.2.8 JWL

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

$$\Pi_i = A_i e^{-R_{1,i}V}\left(\frac{\omega_i}{R_{1,i}V} - 1\right) + B_i e^{-R_{2,i}V}\left(\frac{\omega_i}{R_{2,i}V} - 1\right) - \rho_i \omega_i e_{0,i} \quad (5.45)$$

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

$$\Gamma_{r,i} = \omega_i + 1 \quad (5.46)$$

and

$$\begin{aligned}\delta_{r,i} = A_i e^{-\frac{R_{1,i}\rho_{0,i}}{\rho_i}}&\left[\omega_i\left(\frac{1}{R_{1,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{1,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i} \\ &+ B_i e^{-\frac{R_{2,i}\rho_{0,i}}{\rho_i}}\left[\omega_i\left(\frac{1}{R_{2,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{2,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\omega_i} - e_{0,i} \quad (5.47)\end{aligned}$$

NOTE: The JWL model can also be used to describe the unreacted state if a detonating model is used with the "solidJWL" type. For the solid case, $e_{0,i}$ is calculated so that there is zero pressure at the reference density, and uses an offset pressure, "pRef". Thermal energy is also not considered for the solid case

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
R1	Model coefficient [ ]
R2	Model coefficient [ ]
rho0	Unreacted or reference density [kg m^{-3}]
omega	Grüneisen coefficient [ ]
e0	Offset energy (0 by default for gas) [J/Kg]
pRef	Offset pressure (only for solid, 0 by default) [Pa]

5.2.2.9 CochranChan

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

$$p_{\mathrm{ref},i} = A_i V_i^{1-\mathcal{E}_{1,i}} - B_i V_i^{1-\mathcal{E}_{2,i}} \tag{5.48}$$

The functions used to defined the Mie-Grüneisen equations are

$$\Gamma_i = \Gamma_{0,i} + 1, \tag{5.49}$$

$$\Pi_i = \Gamma_{0,i}\rho_i \Bigg(- \frac{A_i}{(\mathcal{E}_{1,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{1,i}} + \frac{B_i}{(\mathcal{E}_{2,i}-1)\rho_{0,i}} V_i^{1-\mathcal{E}_{2,i}} - e_{0,i} \Bigg) - p_{\mathrm{ref},i} \tag{5.50}$$

and

$$\delta_i = \frac{A_i}{\mathcal{E}_{1,i}} \left[\mathcal{E}_{1,i} V_i^{-\mathcal{E}_{1,i}} \frac{\mathcal{E}_{1,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}} + \frac{B_i}{\mathcal{E}_{2,i}} \left[\mathcal{E}_{2,i} V_i^{-\mathcal{E}_{2,i}} \frac{\mathcal{E}_{2,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}} \right] \frac{1}{\Gamma_{0,i}} - e_{0,i} \tag{5.51}$$

Variable	Description
A	Model coefficient [Pa]
B	Model coefficient [Pa]
E1	Model coefficient [ ]
E2	Model coefficient [ ]
rho0	Reference density [kg/m^3]
Gamma0	Grüneisen coefficient [ ]
e0	Reference energy (0 by default) [J/Kg]

5.2.2.10 DoanNickel

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

$$\Gamma_i = 1 + \Big[0.161 + 0.255e^{\tilde{e}/e_1} f_1 + 0.280e^{\tilde{e}/e_2}(1 - f_1) + 0.137e^{\tilde{e}/e_3} f_2 + 0.050 f_3\Big] \left(\frac{\rho_i}{\rho_{0,i}}\right)^{\alpha(\rho_i,\tilde{e})} \tag{5.52}$$

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

$$\alpha(\rho_i, \tilde{e}) = [0.048 f_1 + 0.032(1 - f_1)(1 - f_2)] \log_{10}(\tilde{e}) + 0.045 f_3 \tag{5.53}$$

with

$$f_1 = \frac{1}{\exp\left(\frac{\tilde{e} - e_{O_2->2O}}{\Delta e_{O_2->2O}}\right) + 1} \tag{5.54}$$

$$e_{O_2->2O} = 8.5 + 0.357 \log_{10}\left(\frac{\rho_i}{\rho_{0,i}}\right) \tag{5.55}$$

$$\Delta e_{O_2->2O} = 0.975 \left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.05} \tag{5.56}$$

$$f_2 = \frac{1}{\exp(\frac{e_{N_2->2N} - \tilde{e}}{\Delta e_{N_2->2N}}) + 1} \tag{5.57}$$

$$e_{N_2->2N} = 45.0 \left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.0157} \tag{5.58}$$

$$\Delta e_{N2->2N} = 4.0 \left(\frac{\rho_i}{\rho_{0,i}}\right)^{0.085} \tag{5.59}$$

$$f_3 = \frac{1}{\exp(\frac{e_e - \tilde{e}}{\Delta e_e}) + 1} \tag{5.60}$$

$$e_e = 160.0 \tag{5.61}$$

and

$$\Delta e_e = 6.0 \tag{5.62}$$

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.

5.2.3 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

$$\Gamma_i = \frac{v_i}{C_v}\frac{\partial p}{\partial T} \tag{5.63}$$

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

$$c_i^2 = v_i^2 \left[\left(\frac{\partial p_i}{\partial T} \right)^2 \frac{T}{C_v} - \frac{\partial p}{\partial v_i} \right] \tag{5.64}$$

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

5.2.3.1 perfectGas

The perfect gas equation of state uses the standard ideal gas pressure relation, however the specific heat ratio is not held constant, allowing for the use of non-constant heeat capacity models.

$$p_i = RT\rho_i \tag{5.65}$$

5.2.3.2 JWLC

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

$$p_i = A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i}{V^{1+\omega_i}} \tag{5.66}$$

the speed of sound if defined

$$c_i^2 = \frac{V^2}{\rho_{0,i}} \left(A_i e^{-R_{1,i}V} + B_i e^{-R_{2,i}V} + \frac{C_i(1+\omega_i)}{V^{2+\omega_i}} \right) \tag{5.67}$$

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]
R1	Model coefficient [ ]
R2	Model coefficient [ ]
rho0	Unreacted or reference density [kg m^{-3}]
omega	Grüneisen coefficient [ ]

5.2.3.3 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.

$$p_i = \frac{RT\rho_i}{1 - b_i\rho_i} \tag{5.68}$$

where b is the co-volume of the fluid.

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

5.2.3.4 BKW

The Becker-Kistiakowsky-Wilson 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.

$$p_i = RT\rho_i \left(1 + xe^{\beta x}\right) \tag{5.69}$$

where

$$x = \frac{\kappa k \rho_i}{W_i (T + \theta)^a} \tag{5.70}$$

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

Variable	Description
beta	Model coefficient [ ]
a	Temperature exponent [ ]
Theta	Model coefficient [K]
kappa	Model coefficient [K^a]
k	Specie co-volume [m^3/kmol]

5.2.3.5 BWR

The Benedict-Webb-Rubin 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

$$p_i = RT\rho_i + \left(B_0 RT - A_0 - \frac{C_0}{T^2}\right)\rho_i^2 + (bRT - a)\rho_i^3 + a\alpha\rho_i^6 + c\frac{\rho_i^3}{T^2}(1 + \gamma\rho_i^2)e^{-\gamma\rho_i^2} \tag{5.71}$$

Variable	Description
A0	Model coefficient
B0	Temperature exponent
C0	Model coefficient
a	Model coefficient
b	Model coefficient
c	Model coefficient
alpha	Model coefficient
gamma	Specific heat ratio [ ]

5.2.3.6 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

$$p_i = p_{\text{ref}} + \frac{1}{\kappa n}\left(\frac{\rho_i}{\rho_{0,i}} - 1\right)^n \tag{5.72}$$

Variable	Description
pRef	Reference pressure [Pa]
rho0	Reference density [kg/m^3]
kappa	Model coefficient ($1/K_0$) [Pa^{-1}]
K0	Bulk modulus (can be used in place of κ) [Pa]
n (or K0)	Model coefficient (default 1) [ ]
Gamma	Grüneisen coefficient [ ]

5.2.4 tabulatedEOS

The tabulatedequation of state can be used to calculate the pressure using the density and temperature. The data is read from a two dimensional tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x", "y", and "f" names are "rho", "T", and "p" respectively.

5.2.4.1 BirchMurnaghan2

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

$$p_i = p_{\text{ref}} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right] \tag{5.73}$$

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

Variable	Description
pRef	Reference pressure [Pa]
rho0	Reference density [kg/m^3]
K0	Bulk modulus [Pa]
Gamma	Grüneisen coefficient [ ]

5.2.4.2 BirchMurnaghan3

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

$$p_i = p_{\text{ref}} + \frac{3K_0}{2K_0'}\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{7/3} - \left(\frac{\rho_i}{\rho_{0,i}}\right)^{5/3}\right] \times \left\{1 + 0.75(K_0' - 4)\left[\left(\frac{\rho_i}{\rho_{0,i}}\right)^{2/3} - 1\right]\right\} \tag{5.74}$$

Variable	Description
pRef	Reference pressure [Pa]
rho0	Reference density [kg/m^3]
K0	Bulk modulus [Pa]
K0Prime	Change in bulk modulus w.r.t. pressure [ ]
Gamma	Grüneisen coefficient [ ]

5.2.5 eTabulated/tabulatedMG

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. See 12.1 for more information on usage.
NOTE: Because of the way the selection of models is done, this option will not show up as an available option if other models are used. Currently only "const", "icoTabulated", and "tabulated" transport models are available.
An example is shown below:

```
phase
{
    type            basic;
    thermoType
    {
        transport   const;
        thermo      eTabulated;
        equationOfState tabulatedMG;
    }
    specie
    {
        molWeight   28.97;
    }
    tranpsport
    {
        mu      0;
        Pr      0;
    }
    thermodynamics
    {
        file    "T.csv";
        mod     ln;     // log(p_i)=p_table

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

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

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

        eMod    ln;     // log(e)=mine+j*deltaE
        nE      40;     // Number of rows
        minE    11.8748;
        deltaE  1.0;
        eIsReal false;
    }
}
```

5.3 Fluid Transport

The transport models are used to define quantities such as viscosity and thermal diffusion. Only Newtonian viscosity models are currently available. All transport models, with the exception of the tabulated methods ported from OpenCFD Ltd. (2018b).

5.3.1 const

Constant values for viscosity (μ) and Prandtl number are used.

$$\kappa = \mu C_p / \mathrm{Pr} \tag{5.75}$$

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

5.3.2 sutherland

Viscosity is calculated using Sutherland's formula for viscosity

$$\mu(T) = A_s \frac{\sqrt{T}}{1 + T_s/T} \tag{5.76}$$

$$\kappa = \mu C_v \left(1.32 + \frac{1.77R}{C_v}\right) \tag{5.77}$$

Variable	Description
Ts	Sutherland temperature [K]
As	Sutherland coefficient [kg/m/s/K$^{1/2}$]

5.3.3 logPolynomial

Transport package using polynomial functions of the natural logarithm of temperature for the natural logarithm of dynamic viscosity and thermal conductivity:

$$\log(\mu(T)) = \sum_{i=0}^{7} \mu_i (\log(T))^i \tag{5.78}$$

$$\log(\kappa(T)) = \sum_{i=0}^{7} \kappa_i (\log(T))^i \tag{5.79}$$

Variable	Description
muCoeffs<8>	Dynamic viscosity coefficients
kappaCoeffs<8>	Conductivity coefficients

5.3.4 polynomial

Transport package using polynomial functions of temperature for the dynamic viscosity and thermal conductivity:

$$\mu(T) = \sum_{i=0}^{7} \mu_i T^i \tag{5.80}$$

$$\kappa(T) = \sum_{i=0}^{7} \kappa_i T^i \tag{5.81}$$

Variable	Description
muCoeffs<8>	Dynamic viscosity coefficients
kappaCoeffs<8>	Conductivity coefficients

5.3.5 WLF

Transport package using the Williams-Landel-Ferry model for viscosity of polymer melts (Williams et al., 1955):

$$\mu(T) = \mu_0 \text{exp}\left(-\frac{C_1(T - T_r)}{C_2 + T - T_r}\right) \tag{5.82}$$

$$\kappa(T) = Cp(T)\frac{\mu(T)}{\text{Pr}} \tag{5.83}$$

Variable	Description
mu0	Reference dynamic viscosity [Pa s]
TRef	Referece temperature [K]
C1	WLF constant [ ]
C2	WLF constant [K]
Pr	Prandtl number [ ]

5.3.6 icoTabulated

The tabulated, incompressible transport model can be used to calculate the viscosity and conductivity using only the temperature. The data is read from two one dimensional tables. See Sec. 12.1 for more information on the

available options for modifiers and interpolation schemes. The ”x” and ”f” names are ”T” and ”mu”/”kappa” respectively.

5.3.7 tabulated

The tabulated transport model can be used to calculate the viscosity and conductivity using the density and temperature. The data is read from two, two dimensional tables. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The ”x”, ”y”, and ”f” names are ”rho”, ”T”, and ”mu”/”kappa” respectively.

5.4 Solid Transport

The transport models are used to define the thermal conductivity of the material. All transport models, with the exception of the tabulated methods are ported from OpenCFD Ltd. (2018b).

5.4.1 constIso

A constant, scalar value (isotropic), is used for for thermal conductivity.

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

5.4.2 constAnIso

A constant, vector value (anistropic), is used for thermal conductivity.

Variable	Description
kappa	Thermal conductivity (vector) [W/m/K]

5.4.3 exponential

Thermal conductivity is calculated using an exponential function

$$\kappa(T) = \kappa_0 \left(\frac{T}{T_{ref}} \right)^{n_0} \tag{5.84}$$

Variable	Description
kappa0	Reference thermal conductivity [W/m/K]
n0	Exponential coefficient [ ]
Tref	Reference temperature [K]

5.4.4 polynomial

Thermal conductivity is calculated using polynomial functions of temperature.

$$\kappa(T) = \sum_{i=0}^{7} \kappa_i T^i \tag{5.85}$$

Variable	Description
kappaCoeffs<8>	Conductivity coefficients

5.4.5 tabulatedIso

The tabulated, isotroptic solid transport model can be used to calculate the conductivity using the temperature. The data is read from a one dimensional table. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x" and "f" names are "T" and "kappa" respectively.

5.4.6 tabulatedAnIso

The tabulated, anisotroptic solid transport model can be used to calculate the conductivity using the temperature. The data is read from a one dimensional table. See Sec. 12.1 for more information on the available options for modifiers and interpolation schemes. The "x" and "f" names are "T" and "kappa" respectively.

5.5 Basic thermodynamic model

The 'basic' 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.

5.6 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
    {
        thermoType
        {
            ...
        }
        ...
    }
    products
    {
        thermoType
        {
            ...
        }
        ...
```

```
    }

    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;
    }
}
```

5.6.1 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 model, 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.

$$\dot{\mathrm{E}} = \frac{\mathrm{E}_0 \rho_i}{\rho_{0,i}} \frac{\mathrm{d}\lambda}{\mathrm{dt}} \tag{5.86}$$

The phase properties are calculated using

$$x_i = \lambda_i^{\ell} x_{r,i} + (1 - \lambda_i^{\ell}) x_{u,i)} \tag{5.87}$$

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 (lamdaPow). 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 cells within the charge with the radii specified are set to full activated when run time reaches 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.

Transport of the reaction progress variable is described by the transport equation

$$\frac{\partial \alpha_i \rho_i \lambda_i}{\partial t} + \boldsymbol{\nabla} \cdot (\alpha_i \rho_i \mathbf{u} \lambda_i) = \alpha_i \rho_i \dot{R}_i \tag{5.88}$$

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)

5.6.1.1 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.

5.6.1.2 linear

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

$$\lambda_{i,j} = \begin{cases} 1 & \text{if } |\mathbf{x} - \mathbf{x}_{\text{det},i,j}| < v_{\text{det},i} t \\ 0 & \text{else} \end{cases} \tag{5.89}$$

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

$$\lambda_i = \max(\lambda_{i,j}) \tag{5.90}$$

Variable	Description
vDet	Speed of propagation within the material [m/s]

5.6.1.3 programmedIgnition

The *programmedIgnition* activation model is slightly more complex than the linear activation model and includes the option to use 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

$$\lambda_{\beta,i} = \frac{1 - \frac{\rho_{0,i}}{\rho_i}}{1 - V_{\text{CJ},i}} \tag{5.91}$$

where $V_{\text{CJ},i}$ is the Chapman-Jougnuet specific volume which is computed using the Chapman-Jouguet pressure. The reference density, $\rho_{0,i}$ is read from the product equation of state. The programmed definition of the reaction progress variable is a modified version of the linear activation, where a cell reacts at a time-rate determined by the detonation velocity and the local grid size

$$\lambda_{\text{programmed},i} = \max(t - t_{\text{ign},j}, 0) \frac{v_{\text{det}}}{1.5dx} \tag{5.92}$$

where 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, $\lambda_i = \lambda_{\text{programmed},i}$; and if the *programmedBeta* burn option is used $\lambda_i = \max(\lambda_{i,\beta}, \lambda_{\text{programmed},i})$.

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

5.6.1.4 pressureBased

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

$$\dot{R}_i = \mathrm{I}\left(\frac{\rho_i}{\rho_{0,i}} - 1 - a\right)^x (1-\lambda)^b + \mathrm{G}_1(1-\lambda)^c\lambda^d p^y + \mathrm{G}_2(1-\lambda)^e\lambda^f p^z \quad (5.93)$$

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 [ ]
G1	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 [ ] (default 0)
maxLambda1	Maximum value of λ that the first step is used [ ] (default 1)
G2	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 [ ] (default 0)
maxLambda2	Maximum value of λ that the second step is used [ ] (default 1)
pScale	Coefficient used to scale pressure [ ] (default is 1)

5.6.1.5 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

$$\dot{R}_i = (1-\lambda)\begin{cases} 0 & T < T_{\text{ign}} \\ \mathrm{d_p^2 A_{low}} \exp(\frac{E_{a,\text{low}}}{RT}) & T_{\text{ign}} \leq T < T_s \\ \mathrm{A_{high}} \exp(\frac{E_{a,\text{high}}}{RT}) & T \geq T_s \end{cases} \tag{5.94}$$

Where the particle diameter is determined by the chosen diameter model (see Sec. 9), and R is the ideal gas constant.

Variable	Description
diameterModel	Diameter model used to determine the particle diameter
ALow	Low activation rate [m^{-2} s^{-1}]
EaLow	Low activation energy [m^2/s^2]
AHigh	High activation rate [s^{-1}]
EaHigh	High activation energy [m^2/s^2]
Ts	Switch temperature [K]
Tign	Minimum temperature of reaction

5.6.2 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.

5.6.2.1 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.

5.6.2.2 constant

A constant amount of afterburn energy is added where $\dot{Q}$ is specified by the user.

Variable	Description
Qdot	Afterburn energy addition rate[J/kg/s]

5.6.2.3 linear

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

$$\dot{Q} = \begin{cases} \frac{Q_0}{\Delta t} & \text{if } t_{\text{start}} < t < t_{\text{end}} \\ 0 & \text{else} \end{cases} \tag{5.95}$$

Variable	Description
Q0	Afterburn energy [J/kg]
tStart	Time energy begins being added [s]
tEnd	Time energy stops being added [s]

5.6.2.4 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

$$\frac{\partial \alpha_i \rho_i c_i}{\partial t} + \boldsymbol{\nabla} \cdot (\alpha_i \rho_i \mathbf{u} c_i) = \alpha_i \rho_i a (1 - c_i)^m (p \times p_{\text{scale}})^n \tag{5.96}$$

and the total afterburn energy is found using

$$\dot{Q} = \frac{\mathrm{d}c_i}{\mathrm{d}t} Q_0 \tag{5.97}$$

Variable	Description
a	Model constant [Pa^{-n}]
m	Reaction progress variable exponent [ ]
n	Pressure exponent [ ]
Q0	Afterburn energy [J/kg]
pMin	Minimum pressure that afterburn occurs [Pa]
pScale	Scale factor for pressure (default is 1) [ ]

5.7 Multicomponent thermodynamic model

The 'multicomponent' fluid thermodynamic model allows for the use of multiple species that use the same equation of state by mass averaging equation of state coefficients of the species. The mass fractions of the species are required as initial conditions (Ydefault can be used to give initial conditions for multiple species). Additionally, a "defaultSpecie" must be provided in the thermodynamic dictionary to specify a "slave" specie

Below is an example of a multicomponent fluid

```
type        multicomponent;
species      (ignitor air);
```

```
defaultSpecie air;
calculateDensity yes;
thermoType
{
    transport   const;
    thermo      eConst;
    equationOfState AbelNobel;
}

ignitor
{
    equationOfState
    {
        b               0.99e-3;
    }
    specie
    {
        molWeight       22.935;
    }
    transport
    {
        mu              1.81e-5;
        Pr              0.7;
    }
    thermodynamics
    {
        Cv          1450;
        Hf          1;
    }
}
air
{
    equationOfState
    {
        b           1.02e-3;
    }
    specie
    {
```

```
        molWeight       28.6689;
    }
    transport
    {
        mu              1.81e-5;
        Pr              0.7;
    }
    thermodynamics
    {
        Cv          1450;
        Hf          0;
    }
}
```

5.8 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

$$p = \frac{\sum_i \alpha_i \xi_i p_i}{\sum_i \alpha_i \xi_i} \tag{5.98}$$

where

$$\xi_i(\rho_i) = \frac{1}{\Gamma_i - 1} \tag{5.99}$$

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

$$c_i = \sqrt{\frac{\sum_i y_i \xi_i c_i^2}{\sum_i \xi_i}} \tag{5.100}$$

with

$$y_i = \frac{\alpha_i \rho_i}{\rho} \tag{5.101}$$

Both the initial energy and temperature are calculated using a root finding method (Newton-Raphson by default) that include the contributions of all phases so the calculated values obey the mixture equations. See 12.3 for additional options.

5.9 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 is set to *yes* in the given equation of state dictionary, then 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.

5.10 Example

An example of a single phase phaseProperties can be seen below

```
mixture
{
    type            basic;
    calculateDensity yes;
    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;
    }
}
```

6 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
- kOmega2006
- 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.

7 Burst patches

The ability to have a "bursting" patch can be used with the burst (external patch), burstCyclic (internal with no AMR), or burstCyclicAMI (general internal). All fields require an additional entry in their dictionaries name *intactField* which is used to specify the boundary condition of the field before bursting. The basic burst patch can be created using any mesh generation technique, however the cyclic variations require using the **createBaffles** utility, standard to OpenFOAM. See Sec. 20.5 for an example of how to create and use the patches.

7.1 Damage models

The method to calculate where burst occurs is runtime-selectable and the inputs are placed in the *constant/polyMesh/boundary* file under the corresponding boundary. This can also be added in either the patch entry within *blockMeshDict* of *snappyHexMeshDict*.

Variable	Description
partialBurst	Can individual face burst (no by default)
useDelta	Us the difference across coupled patches rather than the maximum value (yes by default)

7.1.1 pressure

Only pressure is used to determine if burst occurs.

Variable	Description
pName	Name of the pressure field (p by default)
pRef	Reference pressure (0 by default)
pBurst	Pressure at which burst occurs [Pa]

7.1.2 impulse

Only impulse is used to determine if burst occurs. Because an impulse field must be available while running the simulation, it is recommended to use the "impulse" functionObject (Sec. 15), otherwise some other method of calculting impulse needs to be used.

Variable	Description
impulseName	Name of the impulse field (impulse by default)
impulseBurst	Impulse at which burst occurs

7.1.3 pressureAndImpulse

A combination of the pressure and impulse models are used. If either model results in a burst, then burst occurs. Both sets of inputs are required.

8 Region models

When multiple regions are used, `blastFoam` solvers use global poly patches which do not require any special mesh patches to transfer data between field. Instead, the information is transferred using field specific boundary conditions. The interpolation method, sampled patch, and optional displacement field is specified in the *regionProperties* file. An example of *regionProperties* can be seen below

```
regions
(
    fluid       (air)
    solid       (building)
);

interfaces
(
    air
    {
        air_to_building
        {
            sampleRegion    building;
            samplePatch     building_to_air;
            mappingType     AMI;
            interpolationMethod faceAreaWeight;
        }
    }
    building
    {
        building_to_air
        {
            sampleRegion    air;
            samplePatch     air_to_building;
            displacementField pointD;
        }
    }
);
```

8.1 Boundary conditions

Several boundary conditions have been implemented that make use of the global patches and are described below.

8.1.1 globalMapped

Both vol and point field can use the globalMapped boundary condition which directly maps values from one patch to another where the neighbour patch is specified in *regionProperties*.

Variable	Description
nbrName	Name of field to map from

8.1.2 globalInterpolated

Point field can use the globalInterpolated boundary condition which maps a volume field to a point field where the neighbour patch is specified in *regionProperties*.

Variable	Description
nbrName	Name of field to map from

8.1.3 globalTemperatureCoupled

Using the same methodology as the globalMapped boundary condition, the globalTemperatureCoupled boundary condition computes the temperature of a patch using the neighbour temperature, heat transfer coefficient,and radiation heat flux. Optionally when the entire boundary does not overlap the neighbour patch and unmapped temperature can be specified.

Variable	Description
TNbr	Name of neighbour temperature (T by default)
hNbr	Name of neighbour convective heat transfer coefficient (none by default)
h	Name of convective heat transfer coefficient (none by default)
qrNbr	Name of neighbour radiation heat flux (none by default)
qr	Name of radiation heat flux (T by default)
limitGrad	Is the temperature gradient limited (no by default)
unmappedT	Unmapped temperature (293.15 by default)

9 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.

9.1 constant

A constant diameter is used.

Entry	Description
d	Diameter of the phase [m]

9.2 constantMass

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

$$M = \frac{\pi d_0^3 \rho_0}{6} \tag{9.1}$$

and the true diameter is calculated using

$$d = \left(\frac{6M}{\rho\pi}\right)^{1/3} \tag{9.2}$$

Entry	Description
rho0	Reference density [kg/m^3]
d0	Reference diameter [m]

9.3 reacting

$$\frac{\partial d_i}{\partial t} + \mathbf{u}_i \cdot \boldsymbol{\nabla} d_i = k \tag{9.3}$$

where k is the specified reaction rate (see Sec. 9.4)

Entry	Description
d	Initial diameter [m]
reactionRate	Reaction rate model

9.4 Reaction Rates

Reaction rates are used to give rates based on temperature and/or pressure currently there are two options, pressure based, and Arrhenius rate

9.4.1 pressureBased

The pressure based reaction rate uses a coefficient, pressure scale, and pressure exponent to compute the rate

$$k = \beta \left(\bar{p}\right)^{\alpha} \tag{9.4}$$

and $\bar{p} = p \times p_{scale}$. For example, if the pressure relation is in MPa, $p_{scale} = 1e-6$.

Entry	Description
alpha	Pressure exponent [ ]
beta	Coefficient [$\mathrm{Pa}^{-\alpha}$]
pScale	Pressure scale [ ]

9.4.2 Arrhenius

The rate given by the Arrhenius reaction rate is

$$k = AT^{\beta}\exp\left(\frac{-T_a}{T}\right) \tag{9.5}$$

Entry	Description
A	Rate constant [1/s]
beta	Temperature exponent [ ]
Ta	Activation temperature [K]

10 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. The vector limiters (MinmodV for example) are not used with the MUSCL schemes since the limiting is done for each component of the variables.
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             linear;     //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)      quadraticMUSCL vanLeer; // Internal energy
    reconstruct(speedOfSound) vanLeer;          // Speed of sound
}
```

10.1 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.

10.1.1 HLL

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

$$\mathbf{F}^{HLL} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \frac{S_{nei}\mathbf{F}_{own} - S_{own}\mathbf{F}_{nei} + S_{own}S_{nei}(\mathbf{U}_{nei} - \mathbf{U}_{own})}{S_{nei} - S_{own}} & \text{if } S_{own} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{10.1}$$

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

$$S_{own} = \min(u_{own} - c_{own}, \tilde{u} - \tilde{c}) \tag{10.2}$$

$$S_{nei} = \max(u_{nei} + c_{nei}, \tilde{u} + \tilde{c}) \tag{10.3}$$

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.

10.1.2 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,

$$\mathbf{F}^{HLLC} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \mathbf{F}_{*own} & \text{if } S_{own} \leq 0 \leq S_{*} \\ \mathbf{F}_{*nei} & \text{if } S_{*} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{10.4}$$

The contact wave speed is given by

$$S_{*} = \frac{\rho_{own}u_{own}(S_{own} - U_{own}) + \rho_{nei}u_{nei}(S_{nei} - U_{nei}) + p_{nei} - p_{own}}{\rho_{own}(S_{own} - u_{own}) - \rho_{nei}(S_{nei} - u_{nei})} \tag{10.5}$$

and the pressure in the

$$p_{*K} = \rho_K(S_K - U_K)(S_* - u_K) + p_K \tag{10.6}$$

The final fluxes are determined by

$$\mathbf{F}_{*K}^{HLLC} = \frac{S_*(S_K\mathbf{U}_K - \mathbf{F}_K) + S_K p_{*K}\mathbf{D}_*}{S_K - S_*} \tag{10.7}$$

$$\mathbf{D}_* = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{n} \\ S_* \end{pmatrix} \tag{10.8}$$

10.1.3 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.

$$\mathbf{F}^{HLLC-P} = \mathbf{F}^{HLLC} + \mathbf{\Phi}^P \tag{10.9}$$

$$\mathbf{\Phi}_p = (f-1)\frac{S_{own}S_{nei}}{S_{nei} - S_{own}}\frac{1}{1+|\tilde{M}|}\frac{p_{nei} - p_{own}}{\tilde{c}^2}\begin{pmatrix} 1 \\ \tilde{\mathbf{u}} \\ \frac{1}{2}\tilde{u}^2 \end{pmatrix} \tag{10.10}$$

$$f = \min\left(\frac{p_{own}}{p_{nei}}, \frac{p_{nei}}{p_{own}}\right)^3 \tag{10.11}$$

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

$$p_{**} = \theta p_* + (1-\theta)\frac{p_{own} + p_{nei}}{2} \tag{10.12}$$

$$\theta = \min(M_*, 1) \tag{10.13}$$

$$p_{***} = f p_{**} + (1-f)p_* \tag{10.14}$$

where p_{***} replaces p_{*K} in Eq. (10.7)

10.1.4 AUSM+

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

$$\mathbf{F}^{AUSM+} = 0.5\left[M_* c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] - 0.5\left[|M_*| c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] + \mathbf{F}_p \tag{10.15}$$

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ p_* \mathbf{n} \\ p_* \end{pmatrix} \tag{10.16}$$

The star state variables are written as

$$c_* = 0.5(c_{own} + c_{nei}) \tag{10.17}$$

$$M_* = \mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei} \tag{10.18}$$

and

$$P_* = \mathcal{P}^+_{5,own} p_{own} + \mathcal{P}^-_{5,nei} p_{nei} \tag{10.19}$$

where

$$\mathcal{M}^\pm_1(M) = 0.5(M \pm |M|) \tag{10.20}$$

$$\mathcal{M}^\pm_2(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm 0.25(M \pm 1)^2 & \text{else} \end{cases} \tag{10.21}$$

$$\mathcal{M}^\pm_4(M) = \begin{cases} \mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \mathcal{M}^\pm_2(M)\left[1 \mp 16\beta \mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases} \tag{10.22}$$

$$\mathcal{P}^\pm_5(M) = \begin{cases} \frac{1}{M}\mathcal{M}^\pm_1(M) & \text{if } |M| \geq 1 \\ \pm \mathcal{M}^\pm_2(M)\left[(2 \mp M) - 16\alpha M \mathcal{M}^\mp_2(M)\right] & \text{else} \end{cases} \tag{10.23}$$

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

10.1.5 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

$$M_* = \mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei} + M_p \tag{10.24}$$

and

$$P_* = \mathcal{P}^+_{5,own} p_{own} + \mathcal{P}^-_{5,nei} p_{nei} + P_u \tag{10.25}$$

where

$$M_p = -\frac{K_p}{f_a} \max(1 - \sigma \bar{Ma}^2, 0) \frac{p_{nei} - p_{own}}{\rho_* c_*^2} \tag{10.26}$$

$$P_u = -K_u \mathcal{P}^+_{5,own} \mathcal{P}^-_{5,nei} (\rho_{own} + \rho_{nei}) f_a c_* (u_{nei} - u_{own}) \tag{10.27}$$

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.

$$\bar{Ma}^2 = \sqrt{\frac{u_{own}^2 + u_{nei}^2}{2c_*^2}} \tag{10.28}$$

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

10.1.6 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

$$\mathbf{F}_s^{AUSM+up} = \mathbf{F}_p + \dot{m} \begin{cases} \mathbf{U}_{own} & \text{if } \dot{m} > 0 \\ \mathbf{U}_{nei} & \text{if } \dot{m} \leq 0 \end{cases} \tag{10.29}$$

where

$$\dot{m} = \mathcal{F} c_{1/2} M_{1/2} \begin{cases} \alpha_{own}\rho_{own} & \text{if } M_{1/2} > 0 \\ \alpha_{nei}\rho_{nei} & \text{if } M_{1/2} \leq 0 \end{cases} \tag{10.30}$$

$$\mathcal{F} = \frac{(c_{1/2} - \epsilon)[1 + |M_{1/2}|(1 - \mathcal{G}/2)]}{2} \frac{\max(\alpha_{s,own}, \alpha_{s,nei})}{\alpha_{s,max}} [\alpha_{own}\rho_{own} - \alpha_{nei}\rho_{nei}] \tag{10.31}$$

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

$$\mathcal{G} = \max(2(1 - \mathcal{D}\zeta^2) \tag{10.32}$$

and ζ is defined as

$$\zeta = \begin{cases} \frac{\alpha_s^M - \alpha_{s,crit}}{\alpha_{s,max} - \alpha_{s,crit}} & \text{if } \alpha_s^M > \alpha_{s,crit} \\ 0 & \text{if } \alpha_s^M < \alpha_{s,crit} \end{cases} \tag{10.33}$$

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

$$c_{1/2} = \sqrt{\frac{\alpha_{own}\rho_{own}c_{own}^2 + \alpha_{nei}\rho_{nei}c_{nei}^2}{\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei}}} + \epsilon \tag{10.34}$$

The split Mach number is defined as

$$\begin{aligned} M_{1/2} =& \mathcal{M}_{4,own}^{+} + \mathcal{M}_{4,nei}^{-} \\ & -2\frac{K_p}{f_a}\max(1 - \sigma\bar{M}^2, 0)\frac{P_{tot,nei}-, P_{tot,own}}{(\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei} + \epsilon)c_{1/2}^2} \end{aligned} \tag{10.35}$$

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

$$K_p = 0.25 + 0.75(1 - \mathcal{G}/2) \tag{10.36}$$

$$\sigma = 0.75\sigma/2 \tag{10.37}$$

The pressure is defined as

$$\begin{aligned} P_{tot,1/2} = \\ &-K_u f_a (c_{1/2} - \epsilon)\mathcal{P}^+_{5,own}\mathcal{P}^-_{5,nei}(\alpha_{own}\rho_{own} + \alpha_{nei}\rho_{nei})(u_{nei} - u_{own}) \\ &+\mathcal{P}^+_{5,own}p_{tot,own} + \mathcal{P}^-_{5,nei}p_{tot,nei} \end{aligned} \tag{10.38}$$

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

$$K_u = 0.75 + 0.25(1 - \mathcal{G}/2) \tag{10.39}$$

10.1.7 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

$$a^+ = \max(\max(u_{own} + c_{own}, u_{nei} + c_{nei}), 0) \tag{10.40}$$

$$a^- = \min(\min(u_{own} - c_{own}, u_{nei} - c_{nei}), 0) \tag{10.41}$$

$$a_{own} = \frac{a^+}{a^+ + a^-} \tag{10.42}$$

$$a_{nei} = \frac{a^-}{a^+ + a^-} \tag{10.43}$$

and

$$a = \frac{a^- a^+}{a^+ + a^-} \tag{10.44}$$

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

The volumetric fluxes are written

$$\phi_{own} = u_{own} a_{own} - a \tag{10.45}$$

and

$$\phi_{nei} = u_{nei} a_{nei} + a \tag{10.46}$$

The resulting fluxes are

$$\mathbf{F}^{KT} = (\phi_{own} \mathbf{U}_{own} + \phi_{nei} \mathbf{U}_{nei}) + \mathbf{F}_p \tag{10.47}$$

with

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (a_{own} p_{own} + a_{nei} p_{nei}) \mathbf{n} \\ a(p_{own} - p_{nei}) \end{pmatrix} \tag{10.48}$$

10.2 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 linear;
    ...

    reconstruct(lambda.tnt)  linearMUSCL vanLeer;
    reconstruct(c.tnt)       linearMUSCL 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.

10.3 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 3; // Three stages, 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.

10.3.1 Euler

Standard first order time integration

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.49}$$

10.3.2 RK1SSP

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

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.50}$$

Two stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.51}$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.52}$$

Three stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.53}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.54}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^{(2)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{10.55}$$

10.3.3 RK2

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

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.56}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.57}$$

10.3.4 RK2SSP

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

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.58}$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\left(\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{10.59}$$

Three stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.60}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.61}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\mathbf{U}^n + \frac{2}{3}\mathbf{U}^{(1)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{10.62}$$

Four stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.63}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.64}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^{(2)} - \frac{1}{3}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{10.65}$$

$$\mathbf{U}^{n+1} = \frac{1}{4}\mathbf{U}^n + \frac{3}{4}\mathbf{U}^{(3)} - \frac{1}{4}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(3)} \tag{10.66}$$

10.3.5 RK3SSP

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

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.67}$$

$$\mathbf{U}^{(2)} = \frac{1}{4}\left(3\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{10.68}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\left(\mathbf{U}^n + 2\mathbf{U}^{(2)} - 2\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{10.69}$$

Four stages:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.70}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.71}$$

$$\mathbf{U}^{(3)} = \frac{2}{3}\mathbf{U}^n + \frac{1}{3}\mathbf{U}^{(2)} - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{10.72}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^{(3)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(3)} \tag{10.73}$$

10.3.6 RK4

Standard fourth-order Runge-Kutta method:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{10.74}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{10.75}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{10.76}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \left(\mathbf{F}^n + 2\mathbf{F}^{(1)} + 2\mathbf{F}^{(2)} + \mathbf{F}^{(3)}\right) \tag{10.77}$$

10.3.7 RK4SSP

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

$$\mathbf{U}^{(i)} = \sum_{k=0}^{i-1} \left(a_{ik}\mathbf{U}^{(k)} + \Delta t \beta_{ik} \boldsymbol{\nabla} \cdot \mathbf{F}^{(k)}\right) \tag{10.78}$$

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

11 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-8) all with the option to include dynamic load balancing. The original work ('dynamicRefineBalanceFvMesh' and 'dynamicRefineFvMesh') has been used as the base for the 'adaptiveFvMesh' class. This mesh type has been further developed, specifically for compressible flows, and includes modifications for improved stability with meshes created with **snappyHexMesh**. Additional polyhedral refinement has been ported from foam-extend's and allows for general polygon refinement. Currently balancing is not enabled, but is under development.

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 initially small.

Additionally, the 'movingAdaptiveFvMesh' class is based on 'adaptiveFvMesh', and has all of the same features, but includes the ability to use mesh motion. It is recommended to use the 'velocityLaplacian' motion solver.

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. This is 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
balance	Is dynamic load balancing used (default is yes)
protectedPatches	List of patch names that will not be refined or unrefined (default is none)
nRefinementBufferLayers	Number of layers added to refinement (default is 0)
nUnrefinementBufferLayers	number oy layers added to unrefinement (default is 0)
refine	Use refinement (default is yes)
unrefine	Use unrefinement (default is yes)
refineInterval	Number of time steps between refinement (default is 1)
unrefineInterval	Number of time steps between unrefinement (default is 1)
balanceInterval	Number of time steps between balancing (default is 1)
beginRefine	Time when refinement can begin (default is 0 s)
endRefine	Time when refinement stops (default is great s)
beginUnrefine	Time when unrefinement can begin (default is 0 s)
endUnrefine	Time when unrefinement stops (default is great s)
beginBalance	Time when balancing can begin (default is 0 s)
endBalance	Time when balancing stops (default is great s)
dumpLevel	Write the cell and point level as viewable fields (false by default)

11.1 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. Probe locations are automatically refined up to the maximum level of refinement to remove discontinuities in probe sampling.

Entry	Description
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)
maxRefinement	Maximum number of refinement levels
minDx	Minimum cell size
refineProbes	Are probes refined (yes by default)

11.1.1 fieldValue

The error is set to the value of the given field

$$\epsilon_x = x \tag{11.1}$$

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

Entry	Description
field	Error field

11.1.2 delta

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

$$\epsilon_\delta = |x_{own} - x_{nei}| \tag{11.2}$$

where x is a field chosen by the user, and can be any scalar field stored within the simulation. The subscript own denotes values at the current face, while nei denotes the value at the neighboring face.

Entry	Description
deltaField	Field used to compute error

11.1.3 scaledDelta

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

$$\epsilon_{|\delta|} = \frac{|x_{own} - x_{nei}|}{\max(\min(x_{own}, x_{nei}), x_{min})} \tag{11.3}$$

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

Entry	Description
scaledDeltaField	Field used to compute error
xMin (minValue)	Minimum value to scale difference by (default is small)

11.1.4 Density gradient (densityGradient)

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

$$\epsilon_{\boldsymbol{\nabla}\rho} = \max\left(\frac{|(\boldsymbol{\nabla} l_n \rho)_C - (\boldsymbol{\nabla} l_n \rho)_{own}|}{\alpha_f \rho_C / dl + |\boldsymbol{\nabla} l_n \rho)_{own}|}, \frac{|(\boldsymbol{\nabla} l_n \rho)_C - (\boldsymbol{\nabla} l_n \rho)_{nei}|}{\alpha_f \rho_C / dl + |\boldsymbol{\nabla} l_n \rho)_{nei}|}\right)_f \tag{11.4}$$

with

$$(\boldsymbol{\nabla} l_n \rho)_C = \frac{\rho_{nei} - \rho_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|} \tag{11.5}$$

$$(\boldsymbol{\nabla} l_n \rho)_{nei,own} = (\boldsymbol{\nabla} \rho)_{nei,own} \frac{\mathbf{r}_{nei} - \mathbf{r}_{own}}{|\mathbf{r}_{nei} - \mathbf{r}_{own}|} \tag{11.6}$$

$\mathrm{r}_{own,nei}$ 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.

11.1.5 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.

$$\epsilon_{Lohner} = \frac{|x_{own} - 2x_f + x_{nei}|}{|x_{own} - x_f| + |x_{nei} - x_f| + \varepsilon(|x_{own}| + 2|x_f| + |x_{nei}|)} \tag{11.7}$$

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
epsilon	Scaling coefficient

11.1.6 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. Each error estimator can use different levels of maximum refinement that are specified in their respective dictionaries.

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

// Error is calculated using the density gradient
errorEstimator  multicomponent;
errorEstimators
(
    density
    {
        errorEstimator densityGradient;
        lowerRefineLevel 0.01;
        unrefineLevel    0.01;
        maxRefinement    2;
    }
```

```
    pressure
    {
        errorEstimator  scaledDelta;
        deltaField      p;
        minValue        1e3;

        lowerRefineLevel 0.1;
        unrefineLevel    0.01;
        maxRefinement    2;
    }
    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;
        maxRefinement    1;
    }
);
```

Entry	Description
errorEstimators	List of error estimators to be used

11.2 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 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 calculated non-scalar fields and movingWall

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.

11.3 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;
```

12 Numerics

A collection of numerical methods is provided that are used within the models describe prior, and can also be used for new application and solvers. Many of the methods are only listed and the reader should view the provided references for a more complete description.

12.1 Lookup tables

1D, 2D, and 3D lookup tables are provided, all including support for modified data types, runtime selectable interpolation schemes, and the ability to read or construct the necessary data. The name of the given dictionaries and inputs are dictated by the specific models using the tables, "x" and "f" will be used as place holders for these names. If the dictionary named "xCoeffs" is provided "x" is not used to read the given input (i.e. xMod is mod).

When 1D data is read the field can be constructed with 3 methods: directly read, read from a file given a column, or constructed with a spacing.

Variable	Description
file	Name of a file to read from
delim	Delimiter used to seperate data (',' by default)
xCol	Column of the data (first column by default)
x	Data
xMod	Modifier (none by default)
nX	Number of values
minX	Minimum value
maxX	Maximum value
deltaX	Spacing

12.1.1 Modifiers

Different modifiers can be used to interpolate scalar data in different "spaces", and include

- none
- log10
- pow10 (10^x)
- ln
- exp

Variable	Description
xMod	Modifier for the data

12.1.2 Interpolation schemes

Several interpolation schemes are provided to give the user control over the method which is used to interpolate data including

- floor
- ceil
- linearClamp
- linearExtrapolated
- quadraticClamp
- quadraticExtrapolted
- cubicClamp
- cubicExtrapolated

Clamp methods limit the interpolation to the minimum and maximum values of the provided table, and extrapolated methods will use the closest values to extrapolate the given data out of bounds. Different methods can be used for all directions, and "interpolationScheme" is used to define the default method.

Variable	Description
xInterpolationScheme (interpolationScheme)	Interpolation scheme (linearClamp by default)

12.1.3 1D lookup tables

Data is looked up given an x value. First and second derivatives have been implemented and a reverse lookup can be used for scalar data (find f given x).

12.1.4 2D lookup table

Data is looked up given x and y values. The data can be read from a table or from a list. The default deliter is ",". The table can also be flipped when reading if the input is reversed (y by x) and a check is done on the size when reading. First and second derivatives (pure and mixed) have been implemented and a reverse lookup can be used for scalar data (find x or y given f and y or x).

12.1.5 3D lookup tables

Data is looked up given x, y, and z values. The data can be read from a table or a list. The default delimiters are "," for columns and ";" for rows. The table can also be flipped when reading if the input is reversed (z by y by x) and a check is done on the size when reading. First and second derivatives (pure and mixed) have been implemented.

12.2 Equations

A simple equation class has been implemented to support many of the numerical methods. The equations take and input and return a result.

Equation: A single scalar input is used to return a single value. This can be used as a univariate or multivariate equation.

Univariate equation: A List of scalar inputs are used to return a single value. This can be used as a multivariate equation.

Multivariate equation: A list of scalar inputs are used to return a list of values.

12.3 Root solvers

A collection of root solvers are provided for finding the roots of univariate, scalar equations. Both the x (absolute and relative) and y (absolute only) values are checked for convergence. The available methods are:

Univariate root solvers:

- bisection
- Brent
- falsePoint
- Halley
- NewtonRaphson
- secant
- Steffensen
- step

Multivariate root solvers:

- goodBroyden (Broyden, 1965)
- badBroyden (Kvaalen, 1991)
- NewtonRaphson

Variable	Description
xTolerance(s)	Relative tolerance for the difference in x (1e-6 by default)
xAbsTolerance(s)	Absolute tolerance for the difference in x (1e-6 by default)
yTolerance(s)	Absolute tolerance for the y value (1e-6 by default)
maxSteps	Maximum number of iteration (100 by default)

12.4 Minimization

A collection of minimization methods are providing for finding the minimums of scalar equations. All methods were implemented using Kochenderfer and Wheeler (2019). The available models are

Univariate minimization:

- bisection
- Fibonacci
- goldenRatio
- NewtonRaphson
- quadraticFit
- ShubertPiyavskii
- step

Multivariate minimization:

- gradientDescent
- NelderMead
- particleSwarm

Nelder-Mead options:

Variable	Description
reflectionCoeff	Scale for reflection (1 by default)
expansionCoeff	Scale for expansion (2 by default)
contractionCoeff	Scale for contraction (0.5 by default)

Particle Swarm options:

Variable	Description
narticles	Number of particles used (100 by default)
cLocal	Velocity weight toward the best local point
cGlobal	Velocity weight toward the best global point
vWeight	Weight of previous velocity

General options:

Variable	Description
xTolerance(s)	Absolute convergence tolerance for the independent variables (1e-6 by default)
xRelTolerance(s)	Relative convergence tolerance for the independent variables (1e-6 by default)
yTolerance(s)	Absolute convergence tolerance for the evaluated function (1e-6 by default)
yRelTolerance(s)	Relative convergence tolerance for the evaluated function (1e-6 by default)
maxSteps	Maximum number of iteration (100 by default)
nSamples	Number of pre-sample intervals (0 by default)
normaliseError	Is the relative error checked for convergence (true by default)
tau	Line search (and step) reduction coefficient (0.5 by default)

12.5 Integrators

Numerical integrators have been implemented for arbitrary data types (not only scalars).

Univariate integrators (single input) have the ability to use either adaptive integration or a fixed number of intervals. These methods have also been optimised to reduce the number of function evaluations. The available methods are:

Univariate integrators:

- `Boole`
- `Gaussian (arbitrary number nodes)`
- `midPoint`
- `Simpson13`
- `Simpson38`
- `trapezoidal`

Variable	Description
tolerance	Convergence tolerance (1e-6 by default)
adaptive	Is the integration adaptive (true by default)
maxSplits	Maximum number splits (10 by default)
nIntervals	Number of intervals (10 by default)

Because of the complexity of integrating an arbitrary number of dimensions, only a small subset of the univariate integrators a have been implemented for multiple inputs. These methods also do not have the same optimisation as seen in the univariate case. The available options are:

Multivariate integrators:

- `Gaussian (arbitrary number of nodes)`
- `midPoint`
- `trapezoidal`

Variable	Description
tolerances	Convergence tolerance (1e-6 by default)
adaptive	Is the integration adaptive (true by default)
maxSplits	Maximum number splits (5 by default)

13 Solid models

A port of *solids4foam* has been incorporated into the **blastFSI** solver. These models allow for linear and non-linear deformation using the finite-volume methodolgy described in Cardiff et al. (2018); Cardiff and Demirdzic (2021) and Simo and Hughes (2000) for plasticity.
NOTE: These models are still under development and have not been extensively validated *specifically* for blast.

In addition, if `solids4foam` is available locally, the blastSolids4Foam library can also be compiled to allow use of the models available in the **blastFoam** solver to be used with **solid4Foam** solver.

14 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 required, and it 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 and the optional 'protectedPatches' keyword must be used to specify boundaries which cannot be refined. Checkpointing can also not be used with refinement due to limitations on the mesh.

15 Function objects

In addition to the standard OpenFOAM functionObjects, additional blast specific function objects have been added. The standard function objects from the sampling library are automatically included, but additional libraries can also be included. A list of these libraries can be seen in 18. These can be included using the following *controlDict* entry

```
functions
{
    impulse
    {
        type    impulse;
        libs ("libblastFunctionObject.so");
        // 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;

        ...
    }
}
```

15.1 blastMachNo

The 'blastMachNo' 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)

15.2 dynamicPressure

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

$$\mathbf{p}_{dyn} = \frac{1}{2}\rho_i \mathbf{U}|\mathbf{U}|. \tag{15.1}$$

The 'dynamicPressure' 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)

15.3 fieldMinMax (fieldMax)

The minimum or maximum value at every cell in a domain is 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))
restartOnRestart	Reset the min/max on simulation restart (default is no)
mode	Type of min/max (component, magnitude, componentMag)
minMax	Minimum or maximum (default is max)

15.4 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]

15.5 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)

15.6 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)

15.7 writeTimeList

This function object writes the mesh and all fields 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

15.8 conservedQuantities

The volume integrated value of fields can be computed and printed in the output of the solver. The initial, current, total difference, and relative dif-

ference is printed. Any fields 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 into a domain is not considered.

15.9 blastProbes

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: When refinement and unrefinement occurs, the probe locations will "jump" to the nearest cell resulting in fluctuations when plotted. This can be eliminated by allowing refinement of probes in dynamicMeshDict ('refineProbes') (this is the default option).

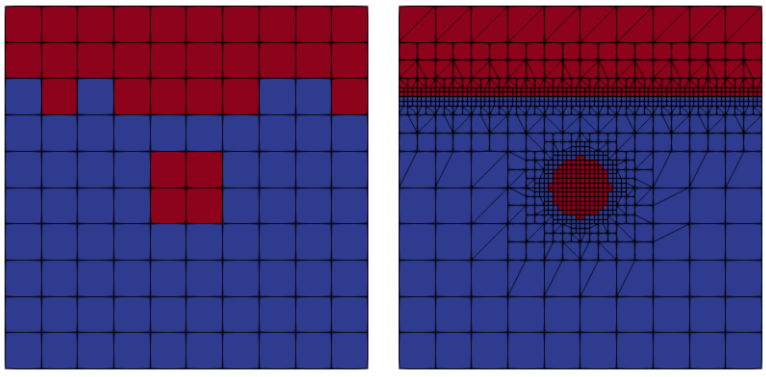
Entry	Description
probeLocations	List of locations to probe
fields	Fields to probe at the given points
append	Are probes appended to and existing file (default no)
fixLocation	Probes follow cell motion (not stationary)
adjustLocations	Can probes be moved into the mesh (default no)
writeVTK	Probed points are written to a VTK file (default no)
interpolationScheme	Method to interpolate probe. Options are cell, cellPoint, cellPointFace, cellPatchConstrained, cellPointWallModified, and pointMVC (default is cell)

16 Utilities

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

16.1 `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 Sec. 11 is present, allowing for 2D and 3D geometries to be set and refined, and the refinement options are specified in *setFieldsDict*, not *dynamicMeshDict*. The utility works by setting the default field values provided, setting the requested fields, then checking for local variations in specific fields. Region may have a level defined where the mesh is refined up to this level. 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 local change in a tracked

field will be refined; however, by specifying an error estimator, the refinement will be determined by the method requested.
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 if non-uniform fields will be used. If balancing is used this will automatically be enabled to ensure that all fields are correct.
In addition to the standard shapes that can be set with **setFields**, several additional options have been added and are listed below with their required entries. All MassToCell options require 'rho' and 'mass' to be specified.

- sphereListToCell: centres, radii, or radius
- cylinderCellToList: p1s, p2s, radii, or radius
- boxMassToCell: centre and aspectRatio (span for backup)
- cylindericalMassToCell: centre, LbyD and direction (radius and L for backup)
- sphericalMassToCell: centre (radius for backup)
- searchableSurfaceToCell: surface (searchable surface type)
- searchableSurfaceToPoint: surface (searchable surface type)

The option to specify non-uniform values for the fields has also been included and are specified by using the field type (i.e. volScalarField, pointVectorField, etc.) followed by the type (i.e. Value, Density, etc.), for example pointScalarFieldValue, volVectorFieldDensity, etc.

- Density: A density is calculated using the given volume integrated quantity, and dividing by the volume of the selected cells
- Function: Values are set based on the specified function.
- InitialValue: The current field is read and saved to ensure the true values are used after refinement.

- MassIntegrated: The mass integrated quantity read and is divided by the mass (density times volume) of the given cell set to determine the field value.
- Sum: The field value is calculated given a summed value and the list of fields to use in the sum
- Value: A uniform value is set

Command line options:

Entry	Description
-forceHex8	Force 3D refinement for compatibility with standard OpenFOAM solvers
-debug	Write intermediate step with error and max cell level
-updateAll	Resize all fields from the current time
-overwrite	Write the mesh to previous location
-noFields	Do not write fields
-noRefine	Do not refine the mesh
-noHistory	Do not write the refinement history (fix the mesh)

Dictionary options:

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 (requires selectionMode)
refinePoints	Cells containing the points refined (requires selectionMode)
nBufferLayer	Number of cells between level boundaries (default is 0)
maxIter	Maximum number of iterations (default is 2*max(level)
errorEstimator	Type of error estimator used (default none)

Field options:

Entry	Description
-setBoundaries ("patches")	Set the boundaries provided
-setAllBoundaries	Set all boundary values
-noInternal	Do not set cell values
-evaluateBoundaries	Correct boundary conditions after setting

Selection modes:

Entry	Description
-all	Select all entries
-internal	Select all entries not on boundaries
-interface	Selected entries between selected and not
-boundary	Select entries on boundaries in the 'patches' entry
-interfaceAndBoundary	Select entries using 'interface' and 'boundary'

Set and zone creation: The functionality of the **topoSet** utility is available in **setRefinedFields** allowing for creation of sets and zones using the refinement shapes. cell/face/point sets/zones are specified in separate lists, and multiple "actions" can be performed on the given sets. The usage is identical to that of **topoSet** and the *tutorials/setRefinedFields* case will offer an extensive list of usage options.
An example *setFieldsDict* is listed below

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

defaultFieldValues
(
    volVectorFieldInitialValue U ( 0 0 0 )
    volScalarFieldFunction p
    {
        // Function3 (i.e. uses x, y, and z)
        type coded;
        code
        #{
            return -x + 2.0*y*z
        #};
    }
    volScalarFieldValue alpha.air 0

    // alpha.water = 1 - (alpha.air + alpha.gas)
    volScalarFieldSum alpha.water 1 (alpha.air alpha.gas)
    volScalarFieldValue rho.air 1.225
);
```

```
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

            // alpha.air = 10 [kg] / sum(V * rho.air) [kg]
            volScalarFieldMassIntegrated alpha.air 10

            // rho.water = 500 [kg]/ Volume
            volScalarFieldDensity rho.water 500.0
        );
    }

    // Refine the faces in the "walls" patch
    patchToFace
    {
        name walls;
        level 2;
        refineFaces yes;
        selectionMode boundary;
    }
```

```
);
```

The tutorial in *tutorials/setRefinedField* details how this an be used as well as the outputs of the *debug* option.

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**. Thus, if it is run in parallel, **setRefinedFields** must be run in parallel after the case has been decomposed.

16.2 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.

16.3 createVTKTimeSeries

This utility simplifies the previous utility by simply creating a *.vtu file which provides the path to 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.

16.4 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.

16.5 initializeAtmosphere

The **initializeAtmosphere** utility is used to initialize stable pressure and density fields such that the hydrostatic pressure gradient and gravitational acceleration force are balanced. Currently, there are two available models: hydrostatic and tabulated. The hydrostatic model dynamically solves for the hydrostatic pressure gradient and adjusts the density accordingly. The tabulated model reads a list of pressures and temperatures versus height to determine the density fields. An optional hydrostatic correction can be used once the table values have been set. See *tutorials/initializeAtmosphere* for an example.
Inputs can either be read from a dictionary or through the command line. If the tabulated option is requested, *atmosphereProperies* must be available.

Entry	Description
-pRef	Reference pressure [Pa]
-hRef	Reference height, or the height at the lowest point of the mesh [m]
-phase	Name of the phase to set (only for multiphase)
-zone	Optional cellZone to set
-type	Optional override of the model. (hydrostatic by default, no default if read from dictionary)

16.6 `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. Only field present in the target case are mapped, however additional fields can be mapped. Because there are no values for

these fields, the closest value on the source mesh is used for cells outside of the mapped cells. Iterative refinement can be used to iteratively rotate fields and refine the mesh using the refinement criteria found in *dynamicMeshDict* allowing for easier initialization when using AMR.
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, map to the closest cell (default is no)
-maxR	Cutoff radius to map (default is a big number)
-centre	Center of rotation on the source mesh (default is the center of the source mesh)
-uniform	Copy uniform fields to the target case (default is no)
-additionalFields	List of additional fields to map
-refine	Iteratively map, rotate and refine the mesh

NOTE: Currently, mapping from decomposed cases is not supported, but mapping to a decomposed case is.

16.7 `convertLagrangianPositions`

Because OpenFOAM has changed the write format of Lagrangian particles to baycentric coordinated in place of a point in physical space, paraview is no longer able to read the positions of particles any more. To alleviate this problem, and allow paraview to read Lagrangian fields (in serial and parallel), the `convertLagrangianPositions` will read in the standard OpenFOAM positions file, convert it to a format that paraview can read and rewrite the *positions* file. If the positions need to be reverted to continue a simulation, the "-revert" optional will rewrite the original positions of the particles.

Entry	Description
-revert	Convert point positions back to original positions

17 fvModels and fvConstraints

In addition to the physics present in the standard solver, additional source terms and models can be included by adding the optional *constant/fvModels* and *system/fvConstraints*. All of these models must have their associated libraries included within the given entries in the files. A list of the libraries and a brief description can be found in Sec. 18

18 Optional libraries

Below is a list of optional libraries to include for functionObject, fvModels, and fvConstraints. When libraries are included additional options will become available in the list of runtime-selectable models.

- "libutilityFunctionObjects.so": General utitilies
- "libfieldFunctionObjects.so": Function objects used for field quantities
- "liblagrangianFunctionObjects.so": Simple lagrangian particles
- "libforces.so": Calculate viscous and pressure forces activing on boundaries
- "libfvConstraints.so": General fvConstraints
- "libfvModels.so": General fvModels
- "liblagrangianParcel.so": Coupled lagrangian particles (one way coupling)

19 Solvers

`blastFoam` offers a wide variety of solvers for simulating transient, highly compressible flows. A short description of each is given below.

19.1 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 phases use the *blastFoam* specific equations of states (see Section 5).

19.2 blastEulerFoam

The **blastEulerFoam** solver is the Euler-Euler (two/multifluid) variant of blastFoam. Currently only fluid-solid simulations are supported; 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 **multiphaseEulerFoam** solver is that any number of particle phases can be used. Currently there are three phase models that can be used: fluid, multiphase, and granular. The fluid phase model is used to describe a single component fluid (such as air). 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 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. 3 and Sec. 4
NOTE: While multiple fluid phases will not cause the solver to fail, theses types of simulations are not currently supported.

19.3 blastFSIFoam

The **blastFSIFoam** solver uses the standard **blastFoam** solver to solve the fluid phase, and the models mentioned in Sec. 13 to handle the solid deformation. The fluid phase uses a moving mesh, while the solid phase can be solved using a total lagrangian or linear (stationary), or updated Lagrangian (moving) method. In order to correctly transfer the information between the patches, the coupledGlobalPolyPatch Cardiff et al. (2018) is used and does not require any changed to the mesh patch types. Fields that require coupling information (temperature, displacment, etc.) should used a global patch type (see Sec. 8.1). If meshes are made for each individual region, the patch types can be specified as usual. However, if the **splitMeshRegions** utility is used to separate a single region mesh into multiple region meshes, the patch names and types may need to be modified
NOTE: This solver is still in development. Please use with caution.

19.4 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.

19.5 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.

19.6 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.

19.7 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. While AMR can be used, dynamic load balancing cannot.

19.8 blastEquation

blastEquation makes use of the numerics library and can either be used to interrogate simple equations, or more complicated ones. This utility can either be used strictly though the command line, or using dictionaries to define equations and the operations they are used with. More control is available when using dictionaries, specifically with respect to root solving, minimization, and integration schemes. Please see *tutorials/blastEquation* for complete examples on how to use this utility.

Entry	Description
-func	Dictionary used to define the equation
-dict	Dictionary used to define operations on the equation
-clean	Remove dynamicCode library
-noClean	Do not remove dynamicCode library
-name	Name of the equation
-fx "pow(x, 3)" "x[0]*x[1]"	Function
-dfdx "3*sqr(x)" "x[1]; x[0]"	First derivative of function (entries seperated by ";" for multiple inputs)
-d2fdx2 "6*x"	Second derivative of function (only for single inputs)
-d3fdx3 "6"	Third derivative of function (only for single inputs)
-P "(c b a)"	Polynomial coefficient ($a + bx + cx^2$)
-eval "x0" or "(x0 ... xn)"	Evaluate at a list of points
-findRoots "(x0 x1 x)"	Find roots between x0 and x1, optionally starting at x
-findAllRoots "(x0 x1)"	Find all roots between x0 and x1
-rootSolver	Root solver
-findPolyRoots	Find roots of a polynomial (only valid with "P")
-minimize "(x0 x1 x)"	Minimize function with bounds x0 and x1, optionally starting at x
-minimizer	Method of minimization
-integrate "(x0 x1)"	Integate between x0 and x1
-integrator	Integration method
-time	Use time (only for cases using meshes)

NOTE: When using command line options, all function definitions and list entries should be in quotes.
NOTE: When multiple inputs are used, "x0" becomes "(x00 ... x0m)", and applies to all entries that take points/positions.

20 Examples

Several validation and tutorial cases are presented to demonstrate the approach for setting up typical problems. Other validation and tutorial cases are available in the *validation/* and *tutorials/* directories, which can be found in the top level directory where `blastFoam` is installed. The cases presented here are

1. A single-phase, double Mach reflection. Located in *validation/blastFoam/doubleMachReflection*
2. A 1-D shock tube consisting of air and water. Located in *validation/blastFoam/shockTube_twoFluid*
3. A 2-D, three-phase case with two different detonating phases where the first explosive material is used to detonate the second charge using the pressure based activation model. Located in *tutorials/blastFoam/twoChargeDetonation*
4. An explosive charge of C-4 buried in sand. Located in `tutorials/blastEulerFoam/mine`

Only modified, relevant input files are shown here, as all additional files (such as `blockMeshDict` and initial conditions) are contained in the respective directory for each case.

20.1 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.

For the double Mach reflection, the boundary conditions must be set so that the fields on the top boundary follow the correct upstream and downstream shock conditions, which are

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

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

These conditions change based on the shock location, leading to a modified boundary condition for density, pressure, temperature, and velocity, which uses a codedFixedValue. 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 uses 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 can be initialized and run using the following commands:

```
blockMesh
setFields
blastFoam
```

The final results are shown in the following figure.

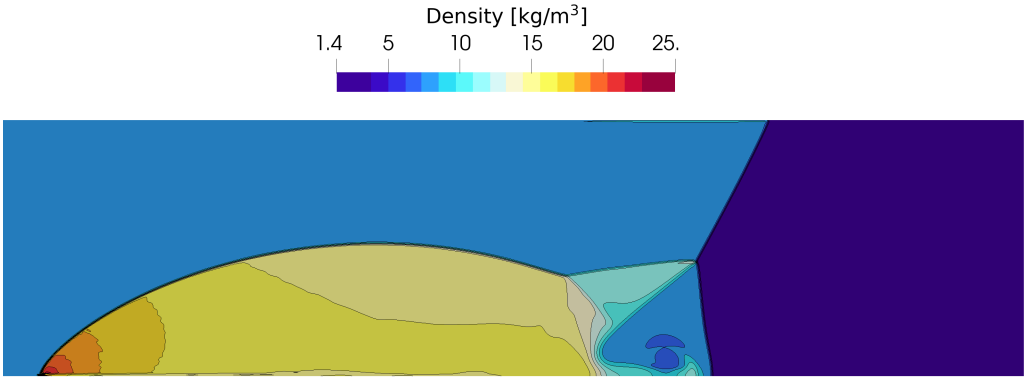


Figure 2: Density contours at t = 200 ms

20.2 Shock Tube - Two Fluid

This 1-D shock tube problem is taken from Zheng et al. (2011) and consists of gas and water initially separated at the center of the domain. The gas

uses the van der Waals EOS and the water uses the Stiffened gas EOS. The shock tube is 1 m long, and is 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:

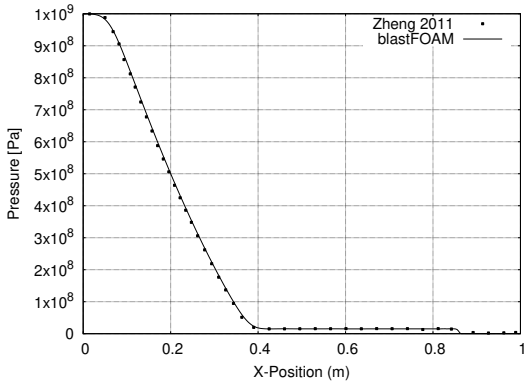
```
fluxScheme      HLLC;

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;
}
```

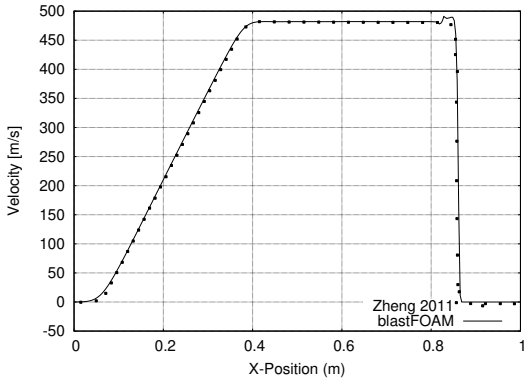
As seen above, the HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 2nd-order strong-stability-preserving time integration. The following commands were used to initialize and run the case in serial:

```
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

20.3 Two charge detonation

This case demonstrates the ability 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 configuration, where the red line is the outline of the RDX charge, the red dot is the initiation point of the RDX charge, and the blue line is the outline of the TNT charge.

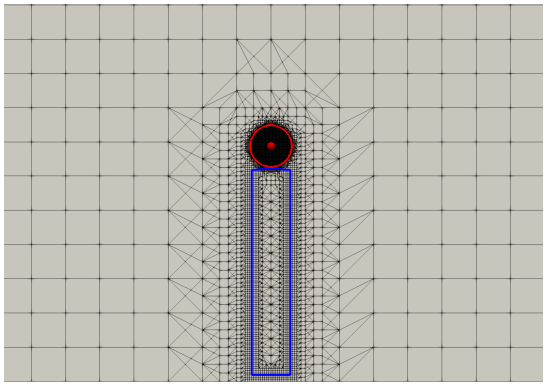


Figure 4: Initial charge configuration

The *phaseProperties* dictionary is defined as

```
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. The *fvSchemes* dictionary is defined as

```
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 defined as

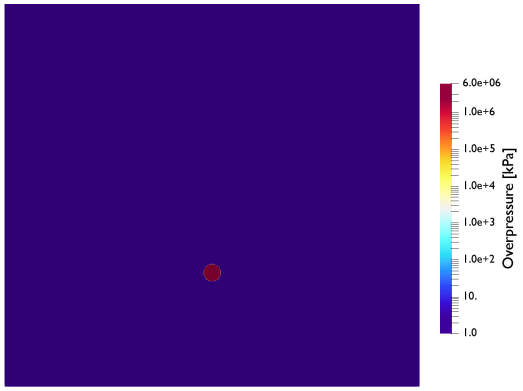
```
fields (alpha.RDX alpha.tnt);
nBufferLayers 1;


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
        );
    }
);
```

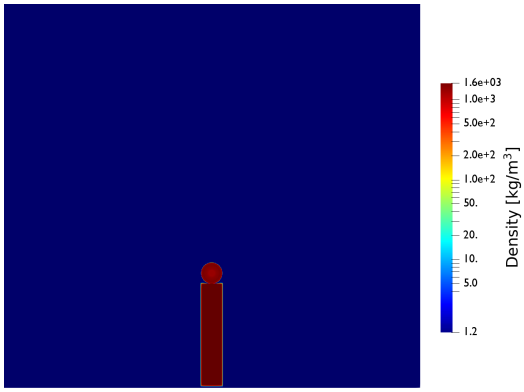
The following commands were used to run the case in parallel:

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

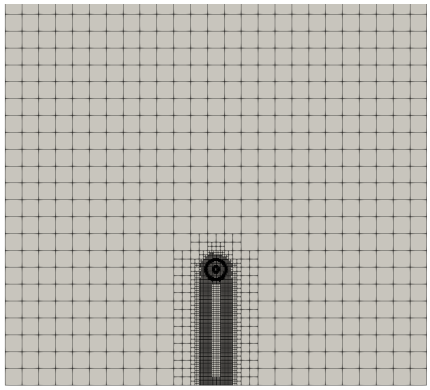
The overpressure, density, and updated mesh are shown in Fig. 5, Fig. 6, and Fig. 7 for time states at 0.005 ms, 0.05 ms, and 0.15 ms, respectively.



(a) Overpressure

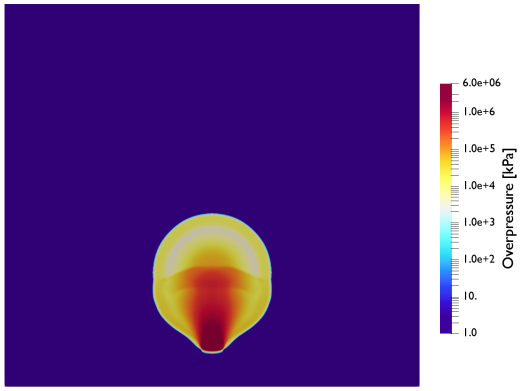


(b) Density

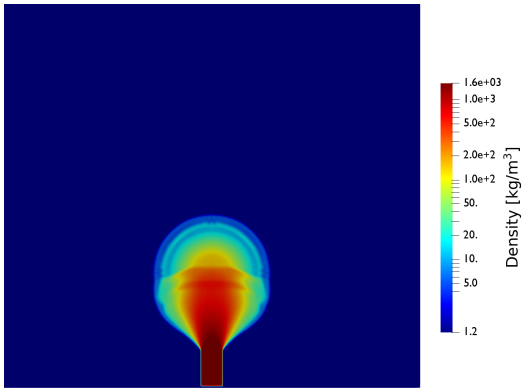


(c) Mesh

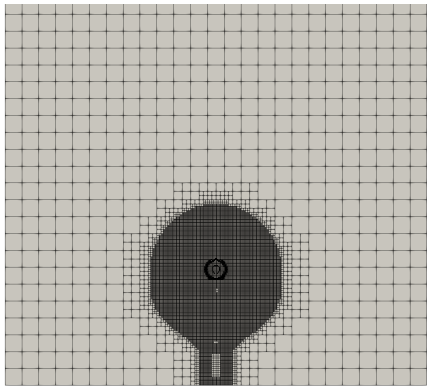
Figure 5: Instantaneous fields at t = 0.005 ms



(a) Overpressure

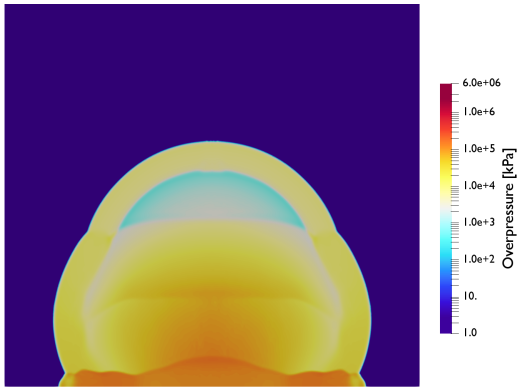


(b) Density

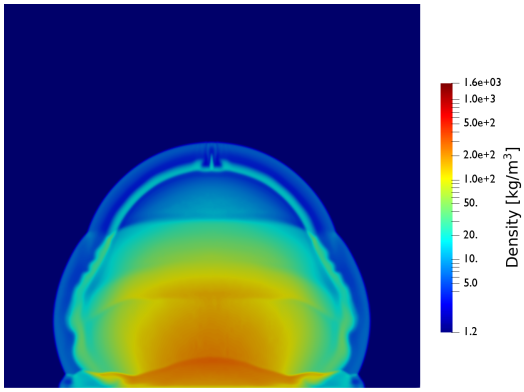


(c) Mesh

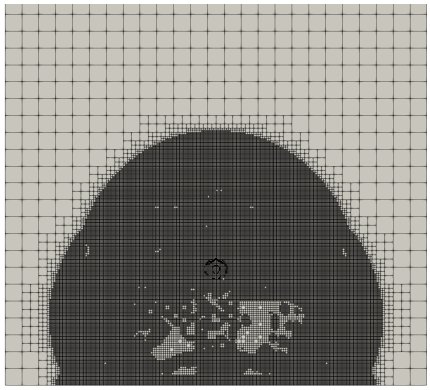
Figure 6: Instantaneous fields at t = 0.05 ms



(a) Overpressure



(b) Density



(c) Mesh

Figure 7: Instantaneous fields at t = 0.15 ms

20.4 Mine (buried charge)

This case demonstrates the simulation of an underground detonation, i.e., a mine. A C-4 charge is buried in a volume of particles (sand) and detonated, leading to the ejection of explosive gases and particles into the air above. The multiphase phase model is used to represent the air and C-4 gases. The particle phase is represented by the granular phase model with a density of 1470 kg/m^3 and a diameter of 50 μm. The *phaseProperties* file is shown 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

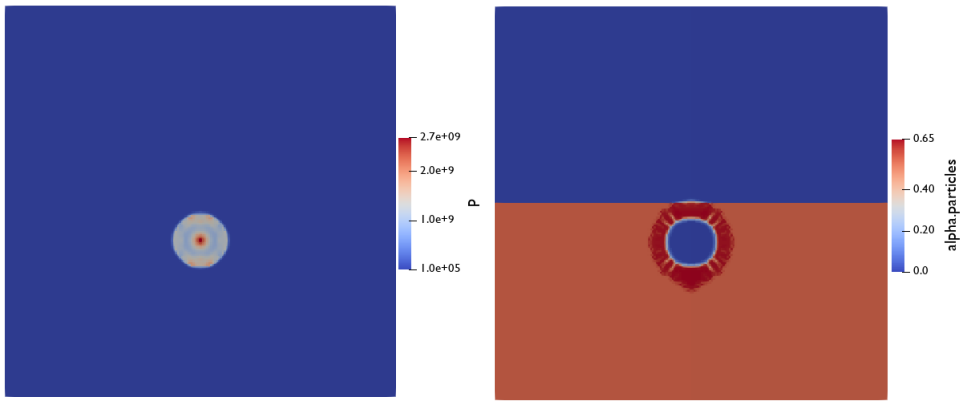
```
fields (alpha.particles);
nBufferLayers 1;

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:

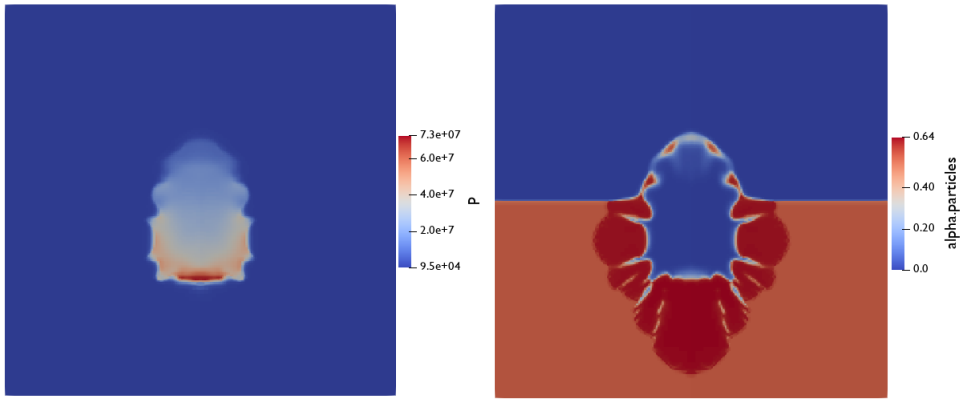
```
blockMesh
decomposePar
mpirun -np $nProcs setRefineFields -parallel
mpirun -np $nProcs blastEulerFoam -parallel
```

Contour plots of the pressure and particle volume fraction are shown in Fig. 8, Fig. 9, and Fig. 10 at time states of 0.0002 s, 0.002 s, and 0.005 s, respectively. Note that the results shown here were mirrored during post-processing; the simulation used an axisymmetric domain.



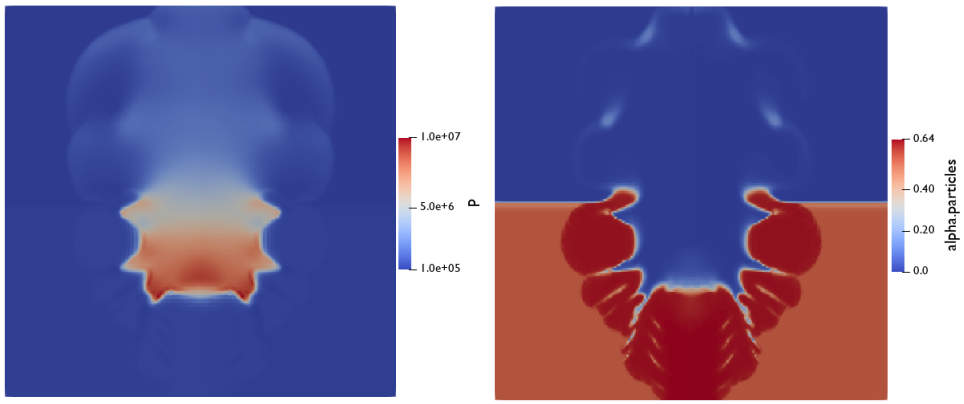
(a) Pressure (b) Particle volume fraction

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

20.5 Bursting windows

This case will demonstrate how to set up a **blastFoam** case to use burst patches. Only the mesh generation and initial conditions will be described since the other parts of the case setup remain the same.

After constructing the initial mesh using block mesh with bounds (-5, 0, 0) to (10, 10, 10) with a symmetry condition for the x-z plane, **snappyHexMesh** is used to cut cells from inside the building and refine the faces on the boundary of the "windows" searchableBox. The *geometry* and *castellatedMeshControls* sections are shown below

```
castellatedMesh on;
snap            on;
addLayers       off;

geometry
{
    building
    {
        type triSurfaceMesh;
        file "building.stl";
        scale 0.001;
    }
```

```
    windows
    {
        type searchableBox;
        min (1.125 -2.375 -1);
        max (5.875 2.375 3.875);
    }
};

castellatedMeshControls
{
    // Refinement parameters
    // ~~~~~~~~~~~~~~~~~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from from refinement followed by balancing
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 10000000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 200000000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 2;

    maxLoadUnBalance 0.1;

    // Number of buffer layers between different levels.
    // 1 means normal 2:1 refinement restriction, larger means slower
    // refinement.
    nCellsBetweenLevels 2;
```

```
    resolveFeatureAngle 30;
    allowFreeStandingZoneFaces false;

    features
    (
        {
            file  "building.extendedFeatureEdgeMesh";
            level 0;
        }
    );

    refinementSurfaces
    {
        building
        {
            level (2 2);
            patchInfo
            {
                type wall;
            }
        }
    }

    refinementRegions
    {
        windows
        {
            mode distance;
            levels ((0.25 2));
        }
    }

    locationInMesh (0 1 1.5);
}
```

Once the mesh has been created with the new building patch, the faces belonging to the widows are selected using **topoSet**. The *topoSetDict* is shown below.

```
actions
(
    {
        name    windows;
        type    faceZoneSet;
        action  new;
        source  searchableSurfaceToFaceZone;
        sourceInfo
        {
            surface searchableBox;
            min (1.125 -2.375 -1);
            max (5.875 2.375 3.875);
        }
    }
);
```

After the necessary faces are selected, the **createBaffles** utility is used to change the faces to boundary patches named windows_master and windows_slave patches with the burstCyclicAMI patch type. The *creatBafflesDict* can be seen below.

```
// Whether to convert internal faces only
// (so leave boundary faces intact).
// This is only relevant if your face selection type can pick
// up boundary faces.
internalFacesOnly true;

// Baffles to create.
baffles
{
    windows
    {
        //- Use surface to select faces and orientation.
        type        faceZone;
        zoneName    windows;

        // Generate patchGroup windows with two patches:
        //  - windows_master
```

```
        //  - windows_slave
        patchPairs
        {
            type            burstCyclicAMI;
            partialBurst    yes;
            pBurst          5e6;
        }
    }
}
```

The initial conditions for the U field can be seen below. The same format will be used will all other initial fields.

```
dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    //- Set patchGroups for constraint patches
    #includeEtc "caseDicts/setConstraintTypes"

    building
    {
        type            zeroGradient;
    }

    windows_master
    {
        type            burstCyclicAMI;
        intactPatch
        {
            type      zeroGradient;
        }
    }
    windows_slave
    {
        type            burstCyclicAMI;
```

```
        intactPatch
        {
            type      zeroGradient;
        }
    }

    outlet
    {
        type            zeroGradient;
    }

    ground
    {
        type            zeroGradient;
    }

    symmetry
    {
        type            symmetry;
    }
}
```

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

```
surfaceFeatures
blockMesh
decomposePar -copyZero
mpirun -np 4 snappyHexMesh -overwrite -parallel
mpirun -np 4 topoSet -parallel
mpirun -np 4 createBaffles -overwrite -parallel
mpirun -np 4 setRefinedFields -parallel
mpirun -np 4 blastFoam -parallel
```

Contour plots of the overpressureare on the building shown in Fig. 11 and Fig. 12 at time states of 0.0003 and 0.001 s respectively, and the threshold filter is used to remove burst faces. Note that the results shown here were mirrored during post-processing.

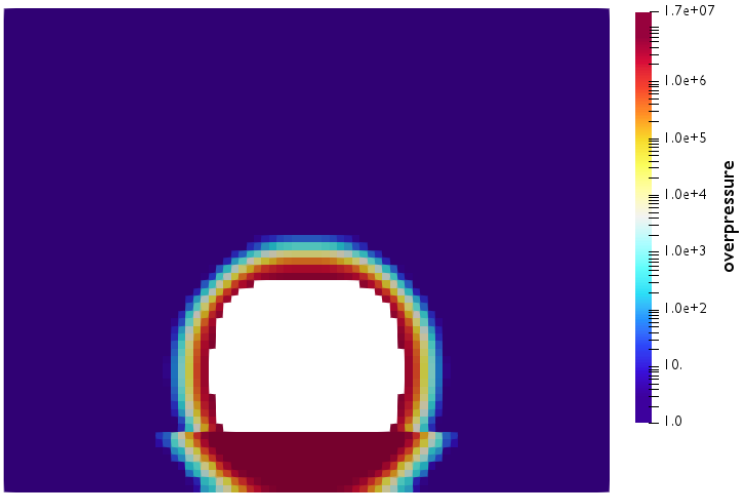


Figure 11: Overpressure field at t = 0.0003 s

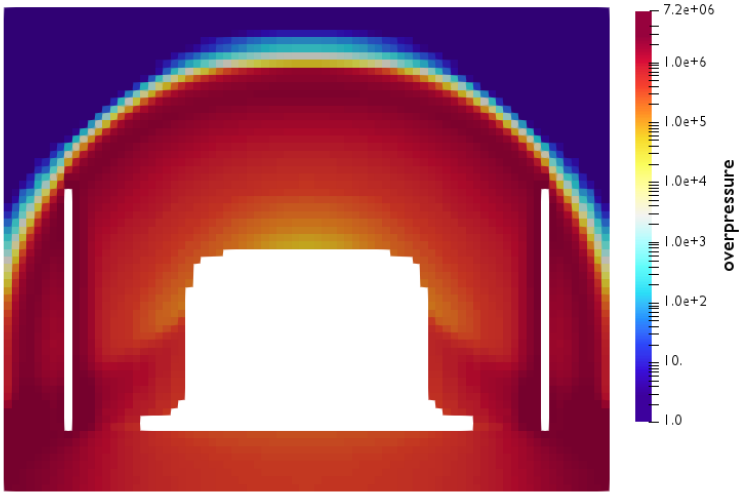


Figure 12: Overpressure field at t = 0.0010 s

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.

Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593.

Cardiff, P. and Demirdzic, I. (2021). Thirty years of the finite volume method for solid mechanics. *Archives of Computational Methods in Engineering*, 28(5):3721–3780. Publisher: Springer.

Cardiff, P., Karac, A., De Jaeger, P., Jasak, H., Nagy, J., Ivankovic, A., and Tukovic, Z. (2018). An open-source finite volume toolbox for solid mechanics and fluid-solid interaction simulations. *arXiv preprint arXiv:1808.10736*.

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.

Dittus, F. W. and Boelter, L. M. K. (1930). *Heat transfer in automobile radiators of the tubular type*. Number v. 2, no. 13 in University of California publications in engineering. University of California press, Berkeley, Calif.

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.

Gelperin, N. I. and Einstein, V. G. (1971). "Heat Transfer in Fluidized Beds. In *Fluidization*. Academic Press, New York.

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: 3XgpHQAA-CAAJ.

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.

Gunn, D. J. (1978). Transfer of heat or mass to particles in fixed and fluidised beds. *International Journal of Heat and Mass Transfer*, 21:467–476.

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.

Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for Optimization*. MIT Press, Cambridge, MA, USA.

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.

Kvaalen, E. (1991). A faster Broyden method. *BIT Numerical Mathematics*, 31(2):369–372.

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 multi-material 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.

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.

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.

Simo, J. C. and Hughes, T. J. R. (2000). *Computational Inelasticity*. Springer New York. Google-Books-ID: ftL2AJL8OPYC.

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.

Tillotson, J. H. (1962). Metallic Equations of State For Hypervelocity Impact. *General Atomic Report GA-3216. 1962. Technical Report.*

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. Technical report, NAVAL SURFACE WARFARE CENTER INDIAN HEAD DIV MD.

Williams, M. L., Landel, R. F., and Ferry, J. D. (1955). The Temperature Dependence of Relaxation Mechanisms in Amorphous Polymers and Other Glass-forming Liquids. *Journal of the American Chemical Society*, 77(14):3701–3707. Publisher: American Chemical Society.

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.

Xue, Q., Heindel, T. J., and Fox, R. O. (2011). A CFD model for biomass fast pyrolysis in fluidized-bed reactors. *Chemical Engineering Science*, 66(11):2440–2452.

Yu, A. B. and Standish, N. (1987). Porosity calculations of multi-component mixtures of spherical particles. *Powder Technology*, 52(3):233–241.

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.