blastFoam Theory and User Guide

Version 2.0.0

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

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

October 30, 2019

Abstract

blastFoam is an open-source toolbox for simulating detonations based on the OpenFOAM®[1] framework [OpenCFD Ltd., 2018]. *blastFoam* provides solutions to the five equation model for multi-phase compressible flow, and is based on the work of Zheng et al. [2011] and Shyue [2001]. *blastFoam* provides implementations of the essential numerical methods (e.g. 2nd and 3rd order schemes), equations of state (e.g. Jones-Wilkins-Lee, Ideal Gas, Stiffened Gas, Tait, Cochran Chan, and Van der Waals), 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 a simplified burn model to simulate activation of energetic materials, and supports multi-point initiation and multi-material detonation; and includes an enhanced implementation of the JWL equation of state suitable for modeling afterburn.

[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 library based on the framework of OpenFOAM [OpenCFD Ltd., 2018] to simulate detonation using the five equation model. Along with the main solver, additional utilities have been added to simplify case setup.

This purpose of this guide is to serve as both a reference on the governing equations of *blastFoam* and the models that have been implemented, as well as to briefly explain the capabilities of the solver as well as new functionalities that has been introduced on top of those available in the standard OpenFOAM-7 [OpenCFD Ltd., 2018].

1.1 System Requirements

blastFoam is currently builds against OpenFOAM-7[2]. Aside from packages required to compile OpenFOAM, and the libraries and headers produced by OpenFOAM-7, no other packages are required.

1.2 Downloading

The un-compiled source code can be obtained at `https://github.com/synthetik-technologies/blastfoam`

1.3 Compilation

Cloning the *blastFoam* source code from `https://github.com/synthetik-technologies/blastfoam` Then run the `./Allwmake` command. Ensure that OpenFOAM-7 has been installed and that the environment has been correctly setup (e.g. that you have run something like `source /path/to/openfoam-7/etc/bashrc`.

1.4 Executable

The application to solve these equations is executed by running the *blastFoam* command. The executables are stored within the `$FOAM_USER_BIN` directory, and can be run from any directory.

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

1.5 Getting Help

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

2 Governing equations

The simulation of a collection of highly compressible materials, each governed by a unique equation of state (EOS), is solved using the five equation model. $\mathbf{U}$ is the vector of conservative variables, volume fraction, mass, momentum, and energy, $\mathbf{F}$ are the fluxes corresponding to the respective conservative variables, and $\mathbf{S}$ is a vector of source terms.

$$\partial_t \mathbf{U} + \nabla \cdot \mathbf{F} = \mathbf{S} \tag{1}$$

$$\mathbf{U} = \begin{pmatrix} \alpha_1 \\ \alpha_1\rho_1 \\ \alpha_2\rho 2 \\ \rho\mathbf{u} \\ \rho E \end{pmatrix} \mathbf{F} = \begin{pmatrix} \alpha_1\mathbf{u} \\ \alpha_1\rho_1\mathbf{u} \\ \alpha_2\rho_2\mathbf{u} \\ \rho\mathbf{u}\otimes\mathbf{u} + p\mathbf{I} \\ (\rho E + p)\mathbf{u} \end{pmatrix} \mathbf{S} = \begin{pmatrix} \alpha_1\nabla\cdot\mathbf{u} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2}$$

where ρ is the mixture density, $\mathbf{u}$ mixture velocity, E total energy, p pressure, and ρ_i and α_i are the density and volume fraction of each phase. When only two phases are used, we define

$$\alpha_2 = 1 - \alpha_1. \tag{3}$$

Additional, the mixture density is found using

$$\rho = \sum_i \alpha_i \rho_i. \tag{4}$$

The five equation model transports individual phase masses and volume fraction, but only considers a mixture velocity, energy, and pressure. This reduces the number of evolution equations that need to be solved.

The pressure is defined using a specified EOS, where the mixture's internal energy, densities, and volume fraction are used to calculate the total pressure (Eq. (5)). The equations of state are cast in Mie-Gruneisen form, based of the work of [Zheng et al., 2011].

$$p_i(\rho_i, e_i, \rho) = (\Gamma(\rho_i) - 1)\rho_i e_i - \Pi(\rho_i) \tag{5}$$

3 Equation of states

The mixture pressure is defined in Mie-Gruneisen form as

$$p = \frac{\rho e}{\sum_i \alpha_i \xi_i} - \frac{\sum_i \alpha_i \xi_i \Pi_i}{\sum_i \alpha_i \xi_i}, \tag{6}$$

where

$$\xi_i(\rho_i) = \frac{1}{\Gamma_i - 1}, \tag{7}$$

and Π_i is a function of the equation of state.

The speed of sound within a given phase is given by

$$c_i = \sqrt{\frac{\sum_i y_i \xi_i c_i^2}{\sum_i \xi_i}} \tag{8}$$

with

$$y_i = \frac{\alpha_i \rho_i}{\rho}, \tag{9}$$

$$c_i^2 = \frac{h_i - \delta_i}{\xi_i}, \tag{10}$$

$$h_i = \frac{\Gamma_i p + \Pi_i}{(\Gamma_i - 1)\rho_i}, \tag{11}$$

and δ_i is specific to the equation of state.

The equation of states are specified in the `MieGruneisenEOSProperties` dictionary using

```
phases (phase1 phase2 phase3);
phase1
{
    type JWL;
    ...
}
phase2
{
    type vanderWaals;
    ...
}
phase3
{
    type idealGas;
    ...
}
```

The names of the phases are defined by the user, and correspond to the required initial fields. When two phases are specified, the identity $\alpha_2 = 1 - \alpha_1$ is used and only one volume fraction is transported. If more than two phases are specified, however, all volume fractions are transported. The required fields are the mixture velocity, mixture pressure, mixture internal energy, and phase specific volume fractions and densities.

NOTE: If the maximum value of internal energy is read as less than or equal to zero, the internal energy is initialized using the selected equation of states and the initial pressure and densities. Because the boundary conditions of internal energy need to be read, it is suggested that "e" field is initialized to zero, and allow for the EOS specific initialization of the internal energy field.

The required entries for all equation of states are

Variable	Description
type	Equation of state model
residualRho	Minimum density
residualAlpha	Minimum volume fraction

The following equations of state in functional Mie-Gruneisen form were presented by Zheng et al. [2011].

3.1 Ideal gas (idealGas)

For an ideal gas, the pressure is defined as

$$p_i = (\gamma_i - 1)\rho_i e_i, \tag{12}$$

The functions needed for Eq. (5) are

$$\Gamma_i = \gamma_i, \tag{13}$$

$$\Pi_i = 0, \tag{14}$$

$$\delta_i = 0. \tag{15}$$

Variable	Description
γ	Specific heat ratio

3.2 Generalized van der Waals gas (vanderWaals)

For a gas described by the generalized van der Waals equation of state, the pressure is defined as

$$p_i = \frac{\gamma_i - 1}{1 - b_i\rho_i}(\rho_i e_i + a_i\rho_i^2) - (a_i\rho_i^2 + c_i), \tag{16}$$

Putting the pressure equation in the form of Eq. (5), we obtain

$$\Gamma_i = \frac{\gamma_i - 1}{1 - b_i\rho_i} + 1, \tag{17}$$

$$\Pi_i = \left[1 - \frac{\gamma_i - 1}{1 - b_i\rho_i}\right] a_i\rho_i^2 + \left[\frac{\gamma_i - 1}{1 - b_i\rho_i} + 1\right] c_i, \tag{18}$$

and

$$\delta_i = -b_i\frac{p_i + a_i\rho_i^2}{\gamma_i - 1} + \left(\frac{1 - b_i\rho_i}{\gamma - 1} - 1\right) 2a_i\rho_i. \tag{19}$$

Variable	Description
a	Attraction between particles
b	Specific volume excluded due to particle volume
c	Reference pressure
γ	Specific heat ratio

3.3 Tait

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

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i(b_i - a_i). \tag{20}$$

where a_i, b_i, and γ_i are material properties. In the form of Eq. (5), we have

$$\Gamma_i = \gamma_i, \tag{21}$$

$$\Pi_i = \gamma_i(b_i - a_i), \tag{22}$$

and

$$\delta_i = 0. \tag{23}$$

Variable	Description
a	Bulk modulus
b	Reference pressure
γ	Compressibility

3.4 Stiffened gas (stiffenedGas)

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

$$p_i = (\gamma_i - 1)\rho_i e_i - \gamma_i a_i, \tag{24}$$

where a_i and γ_i are material properties. The functions needed for Eq. (5) are

$$\Gamma_i = \gamma_i, \tag{25}$$

$$\Pi_i = \gamma_i a_i, \tag{26}$$

$$\delta_i = 0. \tag{27}$$

Variable	Description
a	Reference pressure
γ	Compressibility

3.5 Jones Wilkins Lee (JWL)

The more complicated JWL EOS is often used to define energetic materials, and has a pressure given by

$$p_i^* = A_i e^{-R_{1,i}V}\left(1.0 - \frac{\Gamma_0}{R_{1,i}V}\right) + B_i e^{-R_{2,i}V}\left(1.0 - \frac{\Gamma_0}{R_{2,i}V}\right) + \frac{\Gamma_{0,i}(Q + e\rho_i)}{V} \tag{28}$$

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

$$\Gamma_i = \Gamma_{0,i} + 1, \tag{29}$$

$$\Pi_i = \Gamma_{0,i}\rho_i\left(\frac{A_i}{R_{1,i}\rho_{0,i}}e^{-\frac{R_{1,i}\rho_{0,i}}{\rho_i}} + \frac{B_i}{R_{2,i}\rho_{0,i}}e^{-\frac{R_{2,i}\rho_{0,i}}{\rho_i}}\right) - \frac{\Gamma_{0,i}Q\rho_i}{\rho_{0,i}} - p_{ref,i}, \tag{30}$$

and

$$\delta_i = \tag{31}$$

$$A_i e^{-\frac{R_{1,i}\rho_{0,i}}{\rho_i}}\left[\Gamma_{0,i}\left(\frac{1}{R_{1,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{1,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\Gamma_{0,i}} \tag{32}$$

$$+B_i e^{-\frac{R_{2,i}\rho_{0,i}}{\rho_i}}\left[\Gamma_{0,i}\left(\frac{1}{R_{2,i}\rho_{0,i}} + \frac{1}{\rho_i}\right) - \frac{R_{2,i}\rho_{0,i}}{\rho_i^2}\right]\frac{1}{\Gamma_{0,i}} \tag{33}$$

$$-\frac{Q}{\rho_{0,i}}. \tag{34}$$

where Q is determined by the afterburn model.

Variable	Description
A	Model coefficient (dimensions of pressure)
B	Model coefficient (dimensions of pressure)
R_1	Model coefficient
R_2	Model coefficient
ρ_0	Unreacted or initial density
E_0	Initial detonation energy (used to initialize internal energy)
Γ_0	Model coefficient

3.5.1 Activation Model

A simple model is available to activate the JWL material based on a detonation velocity. This model uses a field, λ_a, to track whether a given cell has been "turned on" yet. It is 0 if the $v_{\text{det}}t$ is less than the distance between the initiation point and the cell center, and 1 otherwise. The pressure used in the calculation of fluxes is then given by

$$p_i = \lambda_a p_i^* + (1 - \lambda_a) p_{\text{ref}} \tag{35}$$

Variable	Description
active	Is the activation model in use
points	List of activation points
speed	Speed of propagation within the material
pRef	Reference pressure used before phase is activated

```
phase1
{
    type JWL;
    ...
    initiation
    {
        active  yes;
        points  ((0 0 0) (0.1 0.1 0.1));
        speed   5000;
        pRef    101298;
    }
```

}

Note: If the initiation dictionary is not found all cells are treated as activated.

3.5.2 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. These models are described below.

No Afterburn (none [default])
No additional energy is added.

Constant Afterburn (constant)
A constant amount of afterburn energy is added

Variable	Description
Q_0	Afterburn energy (dimensions of pressure)

Miller Extension Afterburn Model (Miller)
The model of Miller [1995/ed] uses an evolution equation is used to determine the fraction of the total amount of afterburn energy added to the system due to unburnt material. λ is a fraction that has a value between 0 and 1, with an evolution equation given by

$$\frac{\partial \lambda}{\partial t} = a(1-\lambda)^m p^n, \tag{36}$$

and the total afterburn energy is found using

$$Q = \lambda Q_0, \tag{37}$$

Variable	Description
a	Model constant (units consistent with n)
m	Exponent
n	Exponent
Q_0	Afterburn energy (dimensions of pressure)

3.6 Cochran Chan (CochranChan)

The Cochran Chan EOS can be used to describe solid material, and has a reference pressure given by

$$p_{ref,i} = A_i \left(\frac{\rho_{0,i}}{\rho_i}\right)^{1-\mathcal{E}_{1,i}} - B_i \left(\frac{\rho_{0,i}}{\rho_i}\right)^{1-\mathcal{E}_{2,i}} \tag{38}$$

The functions used in Eq. (5) are given by

$$\Gamma_i = \Gamma_{0,i} + 1, \tag{39}$$

$$\Pi_i = \Gamma_{0,i}\rho_i \left(-\frac{A_i}{(\mathcal{E}_{1,i}-1)\rho_{0,i}} \left(\frac{\rho_{0,i}}{\rho_i}\right)^{1-\mathcal{E}_{1,i}} + \frac{B_i}{(\mathcal{E}_{2,i}-1)\rho_{0,i}} \left(\frac{\rho_{0,i}}{\rho_i}\right)^{1-\mathcal{E}_{2,i}} - e_{0,i}\right) - p_{ref,i}, \tag{40}$$

and

$$\delta_i = \tag{41}$$

$$\frac{A_i}{\mathcal{E}_{1,i}} \left[\mathcal{E}_{1,i} \left(\frac{\rho_{0,i}}{\rho_i}\right)^{-\mathcal{E}_{1,i}} \frac{\mathcal{E}_{1,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}}\right] \frac{1}{\Gamma_{0,i}} \tag{42}$$

$$+\frac{B_i}{\mathcal{E}_{2,i}} \left[\mathcal{E}_{2,i} \left(\frac{\rho_{0,i}}{\rho_i}\right)^{-\mathcal{E}_{2,i}} \frac{\mathcal{E}_{2,i} - \Gamma_{0,i} - 1}{\rho_i} + \frac{\Gamma_{0,i}}{\rho_{0,i}}\right] \frac{1}{\Gamma_{0,i}} \tag{43}$$

$$-e_{0,i}. \tag{44}$$

Variable	Description
A	Model coefficient (dimensions of pressure)
B	Model coefficient (dimensions of pressure)
$\mathcal{E}_1$	Model coefficient
$\mathcal{E}_2$	Model coefficient
ρ_0	Reference density
e_0	Reference energy
Γ_0	Model coefficient

4 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 significantly more expensive than their linear counterparts, and are used in implicit solution methods. 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.

4.1 Riemann Solvers

The currently available Riemann solvers are described below. For all schemes, $\mathbf{n}$ denotes the surface normal vector, and $u = \mathbf{u} \cdot \mathbf{n}$.

4.1.1 HLL

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

$$\mathbf{F}^{HLL} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \frac{S_{nei}\mathbf{F}_{own} - S_{own}\mathbf{F}_{nei} + S_{own}S_{nei}(\mathbf{U}_{nei} - \mathbf{U}_{own})}{S_{nei} - S_{own}} & \text{if } S_{own} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \quad (45)$$

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

$$S_{own} = \min(u_{own} - c_{own}, \tilde{u} - \tilde{c}), \quad (46)$$

$$S_{nei} = \max(u_{nei} + c_{nei}, \tilde{u} + \tilde{c}), \tag{47}$$

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.

4.1.2 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,

$$\mathbf{F}^{HLLC} = \begin{cases} \mathbf{F}_{own} & \text{if } 0 \leq S_{own} \\ \mathbf{F}_{*own} & \text{if } S_{own} \leq 0 \leq S_{*} \\ \mathbf{F}_{*nei} & \text{if } S_{*} \leq 0 \leq S_{nei} \\ \mathbf{F}_{nei} & \text{if } 0 \geq S_{nei} \end{cases} \tag{48}$$

The contact wave speed is given by

$$S_{*} = \frac{\rho_{own} u_{own}(S_{own} - U_{own}) + \rho_{nei} u_{nei}(S_{nei} - U_{nei}) + p_{nei} - p_{own}}{\rho_{own}(S_{own} - u_{own}) - \rho_{nei}(S_{nei} - u_{nei})} \tag{49}$$

and the pressure in the

$$p_{*K} = \rho_K (S_K - U_K)(S_{*} - u_K) + p_K \tag{50}$$

The final fluxes are determined by

$$\mathbf{F}_{*K}^{HLLC} = \frac{S_{*}(S_K \mathbf{U}_K - \mathbf{F}_K) + S_K p_{*K} \mathbf{D}_{*}}{S_K - S_{*}} \tag{51}$$

$$\mathbf{D}_{*} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathbf{n} \\ S_{*} \end{pmatrix}. \tag{52}$$

4.1.3 AUSM+ (AUSMPlus)

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

$$\mathbf{F}^{AUSM+} = 0.5\left[Ma_* c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] - 0.5\left[|Ma_*| c_* (\mathbf{U}_{own} + \mathbf{U}_{nei})\right] + \mathbf{F}_p, \tag{53}$$

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ 0 \\ 0 \\ p_* \mathbf{n} \\ p_* \end{pmatrix}. \tag{54}$$

The star state variables are written as

$$c_* = 0.5(c_{own} + c_{nei}), \tag{55}$$

$$Ma_* = 0.5(\mathcal{M}^+_{4,own} + \mathcal{M}^-_{4,nei}), \tag{56}$$

and

$$P_* = 0.5(\mathcal{P}^+_{5,own} + \mathcal{P}^-_{5,nei}), \tag{57}$$

where

$$\mathcal{M}^\pm_1(Ma) = 0.5(Ma \pm |Ma|), \tag{58}$$

$$\mathcal{M}^\pm_2(Ma) = \begin{cases} \mathcal{M}^\pm_1(Ma) & \text{if } |Ma| \geq 1 \\ \pm 0.25(Ma \pm 1)^2 & else \end{cases}, \tag{59}$$

$$\mathcal{M}^\pm_4(Ma) = \begin{cases} \mathcal{M}^\pm_1(Ma) & \text{if } |Ma| \geq 1 \\ \mathcal{M}^\pm_2(Ma)\left[1 \mp 16\beta \mathcal{M}^\mp_2(Ma)\right] & else \end{cases}, \tag{60}$$

$$\mathcal{P}^\pm_5(Ma) = \begin{cases} \frac{1}{Ma}\mathcal{M}^\pm_1(Ma) & \text{if } |Ma| \geq 1 \\ \pm \mathcal{M}^\pm_2(Ma)\left[(2 \mp Ma) - 16\alpha Ma \mathcal{M}^\mp_2(Ma)\right] & else \end{cases}, \tag{61}$$

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

4.1.4 Tadmor/Kurganov

The Tadmor/Kurganov fluxes were developed by Kurganov and Tadmor [2000]. The *rhoCentalFoam* solver [OpenCFD Ltd., 2018, 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

$$a^+ = max(max(u_{own} + c_{own}, u_{nei} + c_{nei}), 0), \tag{62}$$

$$a^- = min(nin(u_{own} - c_{own}, u_{nei} - c_{nei}), 0), \tag{63}$$

$$a_{own} = \frac{a^+}{a^+ + a^-}, \tag{64}$$

$$a_{nei} = \frac{a^-}{a^+ + a^-}, \tag{65}$$

and

$$a = \frac{a^- a^+}{a^+ + a^-}. \tag{66}$$

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

The volumetric fluxes are written

$$\phi_{own} = u_{own} a_{own} - a, \tag{67}$$

and

$$\phi_{nei} = u_{nei} a_{nei} + a. \tag{68}$$

The resulting fluxes are

$$\mathbf{F}^{KT} = (\phi_{own} \mathbf{U}_{own} + \phi_{nei} \mathbf{U}_{nei}) + \mathbf{F}_p, \tag{69}$$

with

$$\mathbf{F}_p = \begin{pmatrix} 0 \\ 0 \\ 0 \\ (a_{own} p_{own} + a_{nei} p_{nei})\mathbf{n} \\ a(p_{own} - p_{nei}) \end{pmatrix}. \tag{70}$$

4.2 Time integration

The addition of 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. 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.

4.2.1 Euler

Standard first order time integration

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{71}$$

4.2.2 RK2

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

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{72}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{73}$$

4.2.3 RK2-SSP (RK2SSP)

Second-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{74}$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\left(\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right). \tag{75}$$

4.2.4 RK3-SSP (RK3SSP)

Third-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{76}$$

$$\mathbf{U}^{(2)} = \frac{1}{4}\left(3\mathbf{U}^n + \mathbf{U}^{(1)} - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{77}$$

$$\mathbf{U}^{n+1} = \frac{1}{3}\left(3\mathbf{U}^n + 2\mathbf{U}^{(2)} - 2\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)}\right) \tag{78}$$

4.2.5 RK4

Standard fourth-order Runge-Kutta method:

$$\mathbf{U}^{(1)} = \mathbf{U}^n - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^n \tag{79}$$

$$\mathbf{U}^{(2)} = \mathbf{U}^{(1)} - \frac{1}{2}\Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(1)} \tag{80}$$

$$\mathbf{U}^{(3)} = \mathbf{U}^n - \Delta t \boldsymbol{\nabla} \cdot \mathbf{F}^{(2)} \tag{81}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n - \frac{1}{6}\Delta t \boldsymbol{\nabla} \cdot \left(\mathbf{F}^n + 2\mathbf{F}^{(1)} + 2\mathbf{F}^{(2)} + \mathbf{F}^{(3)}\right) \tag{82}$$

4.2.6 RK4-SSP (RK4SSP)

Fourth-order, strong-stability-preserving Runge-Kutta method [Spiteri and Ruuth, 2002]:

$$\mathbf{U}^{(i)} = \sum_{k=0}^{i-1}\left(a_{ik}\mathbf{U}^{(k)} + \Delta t \beta_{ik} \boldsymbol{\nabla} \cdot \mathbf{F}^{(k)}\right) \tag{83}$$

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

5 AdaptiveFvMesh

A modified version of `dynamicRefineFvMesh` which uses octree refinement has been added. The modified version allows for both 2D and 3D adaptive refinement. The 3D refinement is done using the standard OpenFOAM implementation, the authors of the 2D refinement have been included in `hexRef2D.H`. The selection of 2D or 3D refinement is done using the number of solution directions and the user is not required to specify the number of refined dimensions. The refinement criteria is based on the work of Zheng et al. [2008], and uses the density gradient to determine what cell should be refined.

Additionally, the option to begin unrefinement at a designated time (*beginUnrefine*) has been added, and can be useful when several time steps are required for a cell to become activated.

OpenFOAM-7 is currently limited in it's ability to dynamically load balance the adaptive mesh, but this is a topic of continuing study.

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

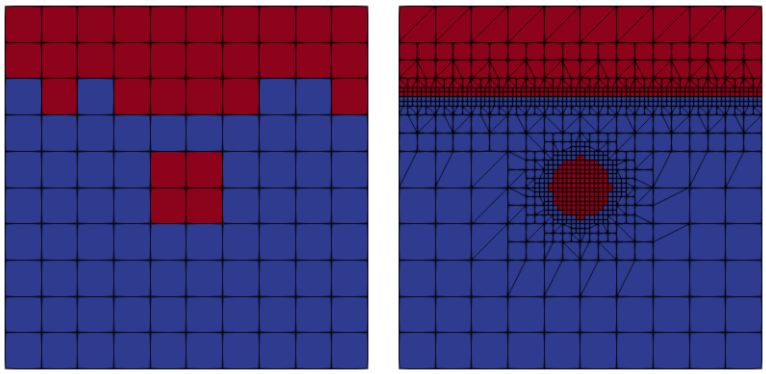
```
dynamicFvMesh    adaptiveFvMesh;

// How often to refine
refineInterval   1;

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

// Field to be refinement on
// (error field is specific to blastFoam)
field            error;

// Refine field in between lower..upper
lowerRefineLevel 1e-2;
upperRefineLevel 1e6;

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

```
// Number of cells between level
nBufferLayers   1;

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

// Stop refinement if maxCells reached
maxCells        1000000;

// Flux field and corresponding velocity field.
// Fluxes on changed faces get recalculated by
// interpolating the velocity.
// Default is none
correctFluxes
(
    (phi none)
    (own none)
    (nei none)
    (alphaPhi.copper none)
    (alphaPhi.gas none)
    (rhoPhi none)
    (alphaRhoPhi.copper none)
    (alphaRhoPhi.gas none)
    (rhoEPhi none)
    (rhoUPhi none)
);

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

6 Pre-Processing

6.1 *setRefinedFields*

This is a modified version of the *setFields* utility in the standard OpenFOAM, however, the ability to refine a *setField* has been added. This is extremely beneficial when you a small volume needs to be set within a set of very large computational cells. The same functionality present in the `adaptiveFvMesh` is present allowing for 2D and 3D geometries to be set. The utility works by setting the default field values, setting the cell or face sets, then checking the difference in one field across all faces. Where residuals occur above a certain user-defined threshold, the mesh is refined. Additionally, by setting the "error" field to a value greater than zero, the entire region it is set in will be refined up to the maximum level, not just the boundaries. The ability to use a coarse base mesh with temporarily refined regions can greatly save on computational expense when simulating very large domains with relatively small areas of interest. 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*

In comparison with *setFields*, the following entries are specific to *setRefinedFields*

Entry	Description
field	Field name that is used to compute error and refine mesh
maxCells	Maximum number of cells allowed
maxRefinement	Maximum level of refinement (every level halves cell size)
nBuffLayer	Number of cells between level boundaries
backup	Dictionary used if a cell set does not contain any cells

An example `setFieldsDict` is listed below

```
field alpha.air;
maxCells 100000;
maxRefinement 3;
nBufferLayers 1;

defaultFieldValues
(
    volVectorFieldValue U ( 0 0 0 )
    volScalarFieldValue p 1.1e5
    volScalarFieldValue alpha.air 0
    volScalarFieldValue rho.air 1
    volScalarFieldValue rho.water 1000
);


regions
(
    cylinderToCell
    {
        p1 (0 0 -1);
        p2 (0 0 1);
        radius 0.1;
        backup
        {
            p1 (0 0 -1);
            p2 (0 0 1);
            radius 0.2;
        }
```

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

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

        fieldValues
        (
            volScalarFieldValue alpha.air 1
        );
    }
);
```

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

7 Example Cases

One validation case and one tutorial case are presented to show the general case setup and run procedure. The first is a simple 1-D shock tube consisting of air and water. The second is a 2D, three-phase case with two different detonating phases consisting of several detonation points. Only the new or modified files will be shown, but the other required files such as `blockMeshDict` and the boundary conditions can be found in the case directories (`validation/shockTube_twoFluid` and `tutorials/twoChargeDetonation`).

7.1 Shock Tube - Two Fluid

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

The left state is defined as:

$$\rho = 1000 \quad \alpha = 1 \quad p = 1e9. \tag{84}$$

The right state is defined as:

$$\rho = 50 \quad \alpha = 0 \quad p = 1e5. \tag{85}$$

The `MieGruneisenEOSProperties` dictionary is

```
phases (fluid gas);

fluid
{
    type            stiffenedGas;
    gamma           4.4;
    a               6.0e8;
    residualRho     1e-10;
    residualAlpha   1e-6;
}

gas
{
```

```
    type            vanderWaals;
    gamma           1.4;
    a               5.0;
    b               1e-3;
    residualRho     1e-10;
    residualAlpha   1e-6;
}
```

and the `fvSchemes` dictionary is

```
fluxScheme      HLLC;

ddtSchemes
{
    default          Euler;
    timeIntegrator   RK2SSP;
}

gradSchemes
{
    default          faceLimited Gauss linear 1;
}

divSchemes
{
    default          none;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default              cubic;
    reconstruct(alpha)   vanLeer 1;
    reconstruct(rho)     vanLeer 1;
    reconstruct(U)       vanLeerV 1;
```

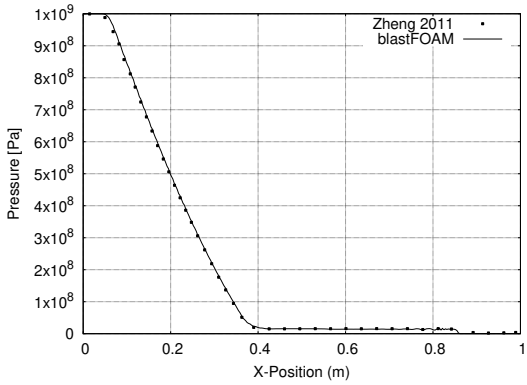
```
    reconstruct(e)      vanLeer 1;
    reconstruct(p)      vanLeer 1;
    reconstruct(c)      vanLeer 1;
}

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.

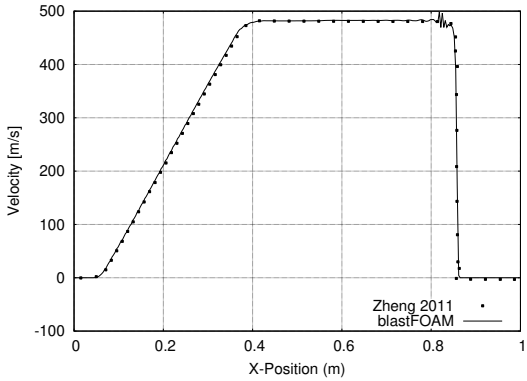
The following commands were used to initialize and run the case in serial

```
blockMesh
setFields
blastFoam
```

The results are shown in Fig. 2.



(a) Pressure



(b) X-velocity

Figure 2: Pressure and velocity fields at t = 73 ms

7.2 Two charge detonation

The final case is meant to show the capabilities of *blastFoam* to solve complex problems involving multiple phases and detonation dynamics. Two charges, one composed of C-4 and the other of TNT are detonated in air. The C-4 has a single detonation point at the center of its circular domain, and the TNT has three equally spaces detonation points within its rectangular domain. Fig. 3 shows the initial charge shapes.

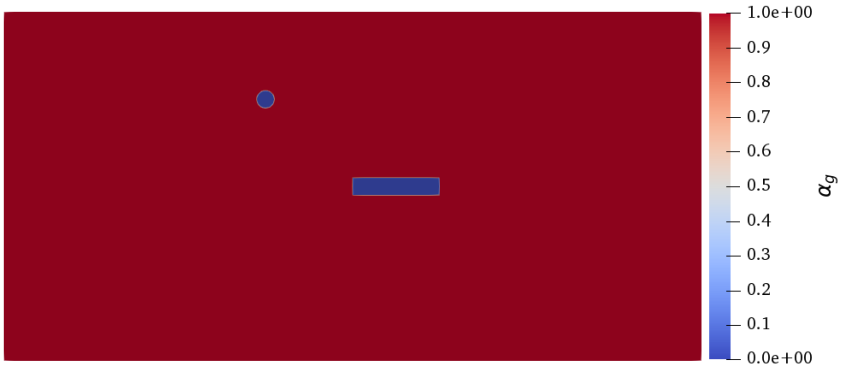


Figure 3: Initial volume fraction of air

The `MieGruneisenEOSProperties` dictionary is

```
phases (c4 tnt gas);

c4
{
    type            JWL;
    A               609.77e9;
    B               12.95e9;
    R1              4.5;
    R2              1.4;
    Gamma0          0.25;
    rho0            1601;
    E0              9e9;
    residualRho     1e-10;
    residualAlpha   1e-6;

    initiation
    {
        active      true;
        points      ((-0.5 0.5 0));
        speed       7850;
        pRef        101298;
    }
}
```

```
tnt
{
    type        JWL;
    rho0        1630;
    A           371.21e9;
    B           3.23e9;
    R1          4.15;
    R2          0.95;
    Gamma0      0.3;
    E0          7e9;
    residualRho 1e-10;
    residualAlpha 1e-6;

    initiation
    {
        active      true;
        points
        (
            (0.125 0 0)
            (0.25 0 0)
            (0.375 0 0)
        );
        speed       7850;
        pRef        101298;
    }
}


gas
{
    type          idealGas;
    gamma         1.4;
    residualAlpha 1e-6;
    residualRho 1e-10;
}
```

the HLLC flux scheme is used with cubic interpolation, vanLeer limiters, and 4th-order strong-stability-preserving time integration. Upwinding is used

for volume fraction in order to ensure that the sum of volume fractions is always equal to one. Due to the fact that all volume fractions are evolved, more complex limiters are required for general flux limiters to be used. This could be the focus of future research. Below is the `fvSchemes` dictionary specifying these.

```
fluxScheme      HLLC;

ddtSchemes
{
    default          Euler;
    timeIntegrator   RK4SSP;
}

gradSchemes
{
    default          faceLimited Gauss linear 1;
}

divSchemes
{
    default          none;
}

laplacianSchemes
{
    default          Gauss linear corrected;
}

interpolationSchemes
{
    default              cubic;
    reconstruct(alpha)   upwind;
    reconstruct(rho)     vanLeer;
    reconstruct(U)       vanLeerV;
    reconstruct(e)       vanLeer;
    reconstruct(p)       vanLeer;
    reconstruct(c)       vanLeer;
```

```
}

snGradSchemes
{
    default           corrected;
}
```

Because the circular 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 is 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

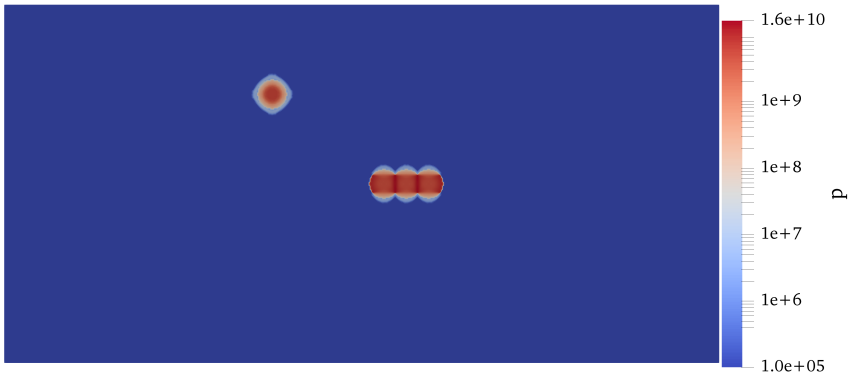
```
field alpha.gas;
maxCells 100000;
maxRefinement 4;
nBufferLayers 2;


defaultFieldValues
(
    volVectorFieldValue U               (0 0 0)
    volScalarFieldValue p               101298
    volScalarFieldValue alpha.gas       1
    volScalarFieldValue alpha.c4        0
    volScalarFieldValue alpha.tnt       0
    volScalarFieldValue rho.gas         1.225
    volScalarFieldValue rho.c4          1601
    volScalarFieldValue rho.tnt         1630
);

regions
(
    cylinderToCell
    {
        p1 (-0.5 0.5 -1);
        p2 (-0.5 0.5 1);
```

```
        radius 0.05;

        backup
        {
            p1 (-0.5 0.5 -1);
            p2 (-0.5 0.5 1);
            radius 0.2;
        }

        fieldValues
        (
            volScalarFieldValue alpha.c4    1
            volScalarFieldValue alpha.gas   0
            volScalarFieldValue error       1
        );
    }
    boxToCell
    {
        box (0 -0.05 -1) (0.5 0.05 1);

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

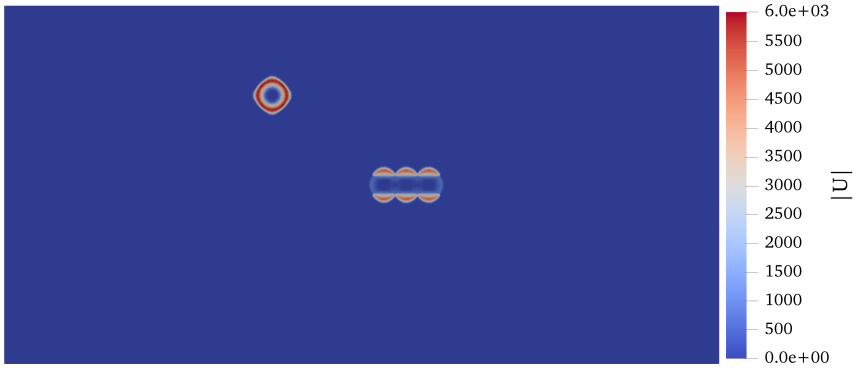
The following commands were used to run the case in parallel

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

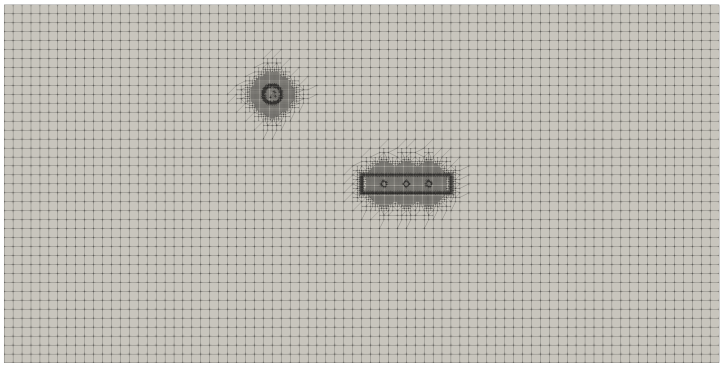
The instantaneous pressure and velocity fields can be seen in Fig. 4 and Fig. 5.



(a) Pressure

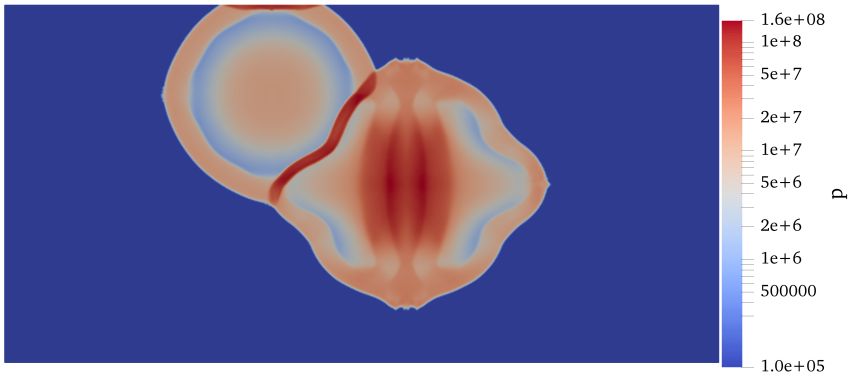


(b) Velocity magnitude

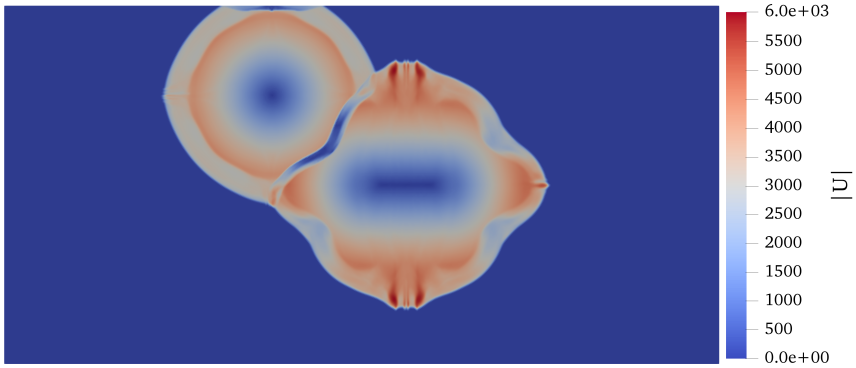


(c) Mesh

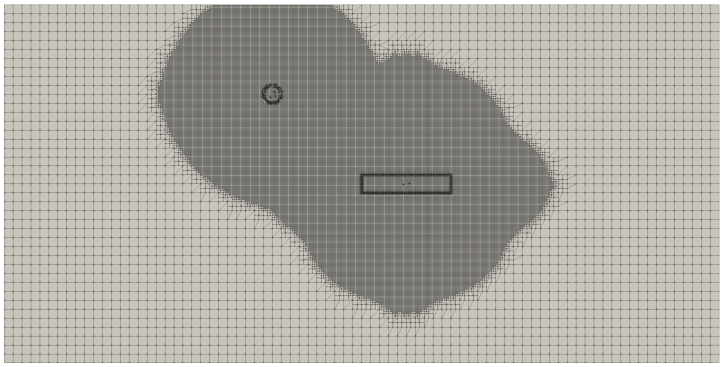
Figure 4: Instantaneous fields at t = 0.01 ms



(a) Pressure



(b) Velocity magnitude



(c) Mesh

Figure 5: Instantaneous fields at t = 0.1 ms

References

Christopher J. Greenshields, Henry G. Weller, Luca Gasparini, and Jason M. Reese. 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, 2010. URL `http://onlinelibrary.wiley.com/doi/10.1002/fld.2069/abstract`.

Alexander Kurganov and Eitan Tadmor. New high-resolution central schemes for nonlinear conservation laws and convection–diffusion equations. *Journal of Computational Physics*, 160(1):241–282, 2000. URL `http://www.sciencedirect.com/science/article/pii/S0021999100964593`.

Hong Luo, Joseph D Baum, and Rainald Löhner. On the computation of multi-material flows using ALE formulation. *Journal of Computational Physics*, 194(1):304–328, February 2004. ISSN 00219991. doi: 10.1016/j.jcp.2003.09.026.

Philip J. Miller. A Reactive Flow Model with Coupled Reaction Kinetics for Detonation and Combustion in Non-Ideal Explosives. *MRS Online Proceedings Library Archive*, 418, 1995/ed. ISSN 0272-9172, 1946-4274. doi: 10.1557/PROC-418-413.

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

Keh-Ming Shyue. A Fluid-Mixture Type Algorithm for Compressible Multicomponent Flow with Mie–Grüneisen Equation of State. *Journal of Computational Physics*, 171(2):678–707, August 2001. ISSN 00219991. doi: 10.1006/jcph.2001.6801.

Raymond J. Spiteri and Steven J. Ruuth. A New Class of Optimal High-Order Strong-Stability-Preserving Time Discretization Methods. *SIAM Journal on Numerical Analysis*, 40(2):469–491, January 2002. ISSN 0036-1429, 1095-7170. doi: 10.1137/S0036142901389025.

Eleuterio F. Toro, Michael Spruce, and William Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock waves*, 4(1):25–34, 1994.

H. W. Zheng, C. Shu, Y. T. Chew, and N. Qin. 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, 2011. ISSN 1097-0363. doi: 10.1002/fld.2380.

H.W. Zheng, C. Shu, and Y.T. Chew. An object-oriented and quadrilateral-mesh based solution adaptive algorithm for compressible multi-fluid flows. *Journal of Computational Physics*, 227(14):6895–6921, July 2008. ISSN 00219991. doi: 10.1016/j.jcp.2008.03.037.