Stony Brook University

Convergent Lagrangian Particle Algorithms for Compressible Fluid Dynamics

A Dissertation Presented

by

Hsin-Chiang Chen

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2015

Stony Brook University

The Graduate School

Hsin-Chiang Chen

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Roman Samulyak – Dissertation Advisor
Professor, Department of Applied Mathematics and Statistics

James Glimm – Chairperson of Defense
Distinguished Professor, Department of Applied Mathematics and Statistics

Yuefan Deng - Member
Professor, Department of Applied Mathematics and Statistics

Robert L. McGraw - Outside Member
Senior Scientist, Biological, Environmental & Climate Sciences Department,
Brookhaven National Laboratory

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Convergent Lagrangian Particle Algorithms for Compressible Fluid Dynamics

by

Hsin-Chiang Chen

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2015

The goal of this thesis is the study of Lagrangian particle methods for complex multiscale hydrodynamic problems. This research has been motivated by difficulties arising in traditional mesh-based methods for the simulation of certain classes of highly non-uniform, complex free surface or multiphase problems. For such problems, Eulerian meshes enhanced with special algorithms for resolving interfaces such as volume-of-fluid, the level set method, arbitrary Lagrangian Eulerian methods, or the method of front tracking which is a hybrid method involving a moving Lagrangian mesh over a fixed Eulerian mesh, are often used. In addition, they require adaptive mesh refinement (AMR). All these methods require complex computationally intensive algorithms for the generation and dynamic adaptation of high quality meshes.

As the method of Smoothed Particle Hydrodynamics (SPH) proposes an attractive alternative to the problems mentioned above, a parallel SPH code has been developed in the 1st phase of the research. The standard SPH algorithms have been enhanced with new implementation of physics models (cavitation, boundary conditions etc.) and applied to the simulation of mercury targets interacting with strong proton pulses in support of the DOE Muon Accelerator Project (MAP). Simulations of MAP experiments that studied splashes of mercury driven by external energy deposition have been performed and good agreement with experimental data has been obtained [2]. But in the course of our work, severe accuracy problems and limitations of SPH have been observed. They confirmed studies published in last years that SPH has zero-convergence order, and is not accurate for many classes of problems.

Motivated by the need to resolve SPH failures while preserving its advantages, we have proposed a new Lagrangian particle method [1, 3] for solving Euler equations for compressible inviscid fluid or gas flows. Similar to smoothed particle hydrodynamics (SPH), the method represents fluid cells with Lagrangian particles. The main features of the method are (1) exact conservation of mass, (2) continuous adaptivity to density changes enabling simulations of large, non-uniform domains, (3) ability to handle material interfaces of any complexity, (4) scalability on modern supercomputers, (5) insignificant increase of algorithmic complexity with increase of spatial dimensionality leading to relatively simple codes in 3D. This also simplifies the portability of codes to new supercomputer architectures. The main contributions of our method, which is different from SPH in all other aspects, are (a) significant improvement of approximation of differential operators based on a polynomial fit and the corresponding weighted least squares problem and convergence of prescribed order, (b) an upwinding second-order particle-based algorithm with limiter, providing accuracy and long term stability, and (c) accurate resolution of states at free interfaces using ghost particles. Numerical verification test demonstrating the convergence order are presented as well as examples of three-dimensional complex free surface flows.

To my father Yu-Shen, Chen, my mother Chin-Kuei, Chou, and my wife Hui-Hui Chang.

Without them nothing would be much worth doing.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Professor Roman Samulyak, who has guided me through my PhD studies and given me insightful advises, without which I will certainly not be able to accomplish the dissertation. I also appreciate all the invaluable discussions with my teammates: Tongfei Guo, Wei Li, Kwangmin Yu, Hyoungkeun Kim, Lina Zhang, and Gaurish Telang. They are my friends and I am glad to have their company at the Stony Brook University. I am very thankful to Professor James Glimm, Professor Robert McGraw, and Professor Yuefang Deng for being on my defense committee. Lastly we want to thank my family for their unconditional love, and my wife for always being there for me - without them I cannot make it through all the difficulties in research and in life.

Chapter 1

Introduction

1.1 Overview and Motivation

High resolution Lagrangian methods are essential for achieving predictive simulations of a wide spectrum of complex free surface/multiphase problems. Most widely used approaches for the simulation of multiphase problems are based on Eulerian meshes enhanced with special algorithms for resolving interfaces such as volume-of-fluid [8], the level set method [9], arbitrary Lagrangian Eulerian methods [10], or the method of front tracking [11] which is a hybrid method involving a moving Lagrangian mesh over a fixed Eulerian mesh. In addition, they often use various adaptive features such as adaptive mesh refinement. These and finite element methods, most common for engineering problems with irregular geometries, require complex computationally intensive methods for the generation of high quality meshes.

Theoretically, the traditional Lagrangian formulation of fluid dynamics [7]

is the basis for the most natural and accurate method for the simulation of complex free surface and multiphase systems, but it suffers from the mesh distortion problem in unsteady, turbulent flows. As a result, the Lagrangian methods are widely used only in 1D for all problems except the dynamics of solids that is characterized by small deformations. The overwhelming majority of solid dynamics codes use finite element-based Lagrangian methods, the fact that speaks for advantages of Lagrangian approaches within their applicability range.

A way to extend the Lagrangian method to 3D was proposed in smoothed particle hydrodynamics (SPH). SPH [14, 15] is a Lagrangian particle method in computational fluid dynamics in which deforming Lagrangian cells are replaced with particles. SPH eliminates the main mesh tangling difficulty of the original Lagrangian method while retaining many of its advantages. Due to its Lagrangian nature, SPH is strictly mass-conservative and capable of robustly handling interfaces of arbitrary complexity in the simulation of free surface and multiphase flows. The representation of matter by particles provides adaptivity to density changes. Not only does this improve the traditional adaptive mesh refinement of structural meshes, that introduces sharp boundaries between mesh patches of different resolution, but also it enables simulations of large ranges of spatial scales (for instance expansion into vacuum and matter islands separated by large vacuum domains). Lastly, there is insignificant increase of algorithmic complexity with increase of spatial dimensionality. This leads to relatively simple codes in 3D and also simplifies the portability of codes to new supercomputer architectures.

To explore the method of Smoothed Particle Hydrodynamics (SPH) as an attractive alternative to to grid-based methods for complex non-uniform problems, a parallel SPH code has been developed in the 1st phase of the research. The standard SPH algorithms have been enhanced with new implementation of physics models (cavitation, boundary conditions etc.) and applied to the simulation of mercury targets interacting with strong proton pulses in support of the DOE Muon Accelerator Project (MAP). Simulations of MAP experiments that studied splashes of mercury driven by external energy deposition have been performed and good agreement with experimental data has been obtained. But in the course of our work, severe accuracy problems and limitations of SPH have been observed. They were especially evident in our implementation of a Poisson problem using SPH discretization of differential operators. Solutions were accurate and second-order convergent only when particles were located on nodes of a rectangular uniform mesh, and became completely incorrect for irregularly placed particles. They confirmed studies published in last years that SPH has zero-convergence order, and is not accurate for many classes of problems.

The major drawback of SPH is a very poor accuracy of discrete differential operators. It is widely accepted [20,21], including original SPH developers [15], that the traditional SPH discretization has zeroth-order convergence for widely used kernels. In addition, the SPH discretization of derivatives is convergent to the order consistent with the interpolating polynomial for the kernel function only if particles are located on a rectangular mesh (which is not the case for unsteady flows). The reason why SPH produces stable and reasonable results

for certain problems, despite using inaccurate and non-convergent discretization of differential operators, is its connection to the Lagrangian/Hamiltonian dynamics of particles [22]. In particular, the traditional discrete SPH equations for the compressible Euler equations are not accurate, but they accurately represent equations of the Lagrangian dynamics of particles interacting via isentropic potentials. The Lagrangian and Hamiltonian properties are also responsible for the long term stability of the traditional SPH. But the Hamiltonian dynamics of particles only approximately represent the dynamics of continuum hydrodynamic systems, and the isentropic interaction energy places additional restrictions.

A number of 'modern' or 'corrected' SPH methods have been developed in recent years (see [20] and reviews [15, 21]). They include the moving-least-squares SPH, 'Godunov'-SPH, P-SPH, PHANTOM etc. But they all improve certain features of SPH at the expense of other properties such as conservation, long-term stability, or prohibitively large number of neighbors that causes other problems. They all still have zero-convergence order, except for the 1st order convergent, moving-least-squates SPH [20], that suffers from long-term stability and other issues.

We have proposed a new Lagrangian particle method for solving compressible Euler equations that eliminates major deficiencies of SPH. First, the presence of large linear errors in SPH differential operators is significantly improved by the local polynomial fitting technique. Second, long term stability is obtained using the upwinding discretization methods. Third, states at free interfaces are accurately resolved using ghost particles. Last and fundamentally

different from SPH in most of approximation, our method is easily generalizable to coupled system of hyperbolic and elliptic or parabolic PDE's for other physics processes.

In the proposed Lagrangian particle method, approximations of spatial derivatives are obtained by employing a local polynomial fit known also as the generalized finite difference (GFD) method [25]. The main idea is to find closest neighbors of each particle and approximate the spatial derivative of a certain physical quantity around the particle location as a linear combination of this quantity at neighboring particles. The optimal coefficients in this linear combination are calculated by solving a least squares problem. Second order accurate spatial discretization is used in the current algorithm, but the GFD method makes it possible to use higher order discretizations with increased particle neighborhoods. Our algorithms uses much smaller number of neighbor particles compared to the Godunov-SPH or other recent SPH modifications that may require hundreds of particles [21].

The conservative Lagrangian formulation of Euler equations is transformed into a quasi-linear form, and an upwinding scheme is employed for the numerical integration. Multiple partial dimensions are resolved using a Strang splitting method for Euler equations. Research on algorithms for elliptic problems involving geometrically complex boundaries and interfaces is in progress [4, 5]. Together with hyperbolic solvers, they form the basis for simulations of complex multiphysics and multiphase systems. The Lagrangian particle method has been implemented in all dimensions, and the 2D and 3D versions have been parallelized using share-memory parallelization techniques and the OpenMP

API. Several 1D, 2D, and 3D simulations are run for accuracy tests, verification of order of convergence, and validation of the capabilities for solving free surface problems. A ghost particle method has been developed for free surface problems. Currently, the Lagrangian particle code is accurate for complex free surface problem in weakly compressible regime (fluids and gases with moderate density changes). It still some additional development in the case of strong compressibility when the density of matter changes by many orders of magnitude. This work is in progress and will be completed in the near future.

1.2 Dissertation Organization

The dissertation is organized as follows. Chapter 2 focuses on SPH. Section 2.1 introduces SPH integral and summation approximations. Section 2.2 describes SPH approximations of Euler equations and main algorithms, such as time integration schemes, neighbour search algorithms, and the treatment of solid boundaries. Section 2.3 demonstrates the capabilities of SPH for nonlinear problems based on the results of the simulations of mercury thimble splash and the jet splash for Neutrino Factory and Muon Collider. Section 2.4 discusses the limitations of SPH, espacially the accuracy of different forms of SPH approximations of the differential operators.

In Chapter 3, We focus on the proposed Lagrangian particle method. Section 3.1 introduces the governing Euler equation in Lagrangian form, and the quasi-linear form. Section 3.2 elaborates on the numerical discretizations and main algorithms of the Lagrangian particle dynamics. Section 3.3 offers an

overview of the developed code for the proposed algorithm.

Section 3.4 gives numerical results demonstrating the accuracy, the order of convergence, and the capability of solving free surface problems, of the proposed algorithm.

In the last chapter we conclude out work and give our perspectives for the future work.

Chapter 2

Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics was introduced by [12] and [13] for the simulations in astrophysics. Due to many of its advantages, SPH has been widely applied to diverse fields in fluid dynamics, geophysics, engineering, and in the film and computer games industry.

SPH is a Lagrangian technique that discretizes the continuum into small elements called the particles, and solves the underlying governing equations based on these particles. In mathematical terms, these particles are interpolation points on which fluid properties can be calculated. In physics terms they are matrial particles in the particle system that carries their own physical states and locations in the Cartessian coordinates.

Unlike traditional mesh-based methods such as finite-difference methods, SPH calculates the spatial derivatives based on analytical differentiation of

the interpolation formula and does not need a grid (see sections 2.1.1 and 2.1.2). This makes diferent forms of SPH approximations of the spatial derivatives possible (see section 2.1.3). Based on the approximations of the spatial derivatives, different forms of SPH approximations of the equations of mass, momentum, and energy can be obtained and are are simply sets of ordinary differential equations (see section 2.2.2).

Since SPH depends use particles and there is no connectivity between any two particles, it adds little complexity when generalizing to higher dimensions, or generalizing to very complex geometries. This is a very important motivation for SPH: the following quote is from the original developer of SPH (from [14]):

> *Although very accurate finite-difference methods exist - and these are better than SPH for some problems - they cannot handle complex physics in three dimensions with the same ease.*

This ease of solving complex physics in 3D makes particle-based methods such as SPH the best candidate for free surface or multiphase problems. However, there are some limitations of SPH as described in section 2.5. This motivates us to develop the new Lagrangian particle method which will be introduced in chapter 3.

In the following sections in this chapter, we will describe the theory of SPH which leads to the approximations of Lagrangian fluid equations, and some related numerical algorithms such as the time integration and the scheme for moving partciles. All these can be found in the SPH literature and for a

comprehensive review of SPH, see [14], [15], [16], [22], [32], and the referrences therein.

We have developed a model for cavitation in section 2.2.6 for the simulations of mercury splish from thimble (see section 2.4.1) and mercury jet experiments for neutrino factory and muon collider (see section 2.4.2). We have also developed a SPH code which is parallelized using MPI (a dirtributed memory parallelization API). The SPH code will be described in section 2.3.

2.1 SPH Formulation

2.1.1 SPH Approximation of A Field Function

SPH expresses each of the fluid dynamic variables as an integral interpolant. This allows any function to be expressed in terms of its values at a set of regularly or irregularly-placed points, that is, the particles. Consider any function $A(\mathbf{r})$ expressed in the form

$$A(\mathbf{r}) = \int A(\mathbf{r}')\,\delta(\mathbf{r}-\mathbf{r}')\,\mathbf{dr}' \tag{2.1}$$

where the integration is taken over the entire three-dimensional space and $\delta(\mathbf{r}-\mathbf{r}')$ is the Dirac delta function. Note that eq. (2.1) is exact but not practically useful. In SPH, the integral interpolant of any function $A(\mathbf{r})$ is defined as

$$A_I(\mathbf{r}) = \int A(\mathbf{r}')\,W(\mathbf{r}-\mathbf{r}', h)\,\mathbf{dr}' \tag{2.2}$$

, where $W(\mathbf{r}-\mathbf{r}', h)$ is known as the *smoothing kernel*, and h is the *smoothing length*, which determines the spatial extent over which W smoothes the variable. Note that W must satisfy two properties,

$$\int W(\mathbf{r}-\mathbf{r}', h)\, \mathbf{dr}' = 1 \tag{2.3}$$

and

$$\lim_{h\to 0} W(\mathbf{r}-\mathbf{r}', h) = \delta(\mathbf{r}-\mathbf{r}') \tag{2.4}$$

, such that $\lim_{h\to 0} A_I(\mathbf{r}) = A(\mathbf{r})$. In the original work of [13], a Gaussian kernel is used. However, instead of the Gaussian kernel function, which has infinite support, kernels with compact support, such as those based on spline functions, are often used for computational efficiency. For example, [17] developed a kernel function based on cubic spline functions:

$$W(q,h) = \frac{\sigma}{h^v}\begin{cases} 1-\frac{3}{2}q^2+\frac{3}{4}q^3 & \text{if } 0\le q\le 1 \\ \frac{1}{4}(2-q)^3 & \text{if } 1\le q\le 2 \\ 0 & \text{if } q>2 \end{cases} \tag{2.5}$$

where $q = \frac{|\mathbf{r}-\mathbf{r}'|}{h}$, v is the number of dimensions, and σ is a normalization constant with values $\frac{2}{3}$, $\frac{10}{7\pi}$, and $\frac{1}{\pi}$ in one, two, and three dimensions, respectively. This kernel has compact support; the second derivative is continuous; and the dominant error term in the integral interpolant (i.e., equation (2.2)) is $O(h^2)$. The choice of the kernel function is instrumental for the success of SPH. Basically, kernel functions should satisfy conditions such as positiv-

ity, compact support, unity, and monotonicity. Details of the construction of kernel functions can be found in Chapter 3 of [32].

The integral interpolant is then approximated by summation interpolant in SPH:

$$A_s(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r} - \mathbf{r}_b, h) \tag{2.6}$$

where the summation is taken over all particles b. Particle b has mass m_b, mass density ρ_b, velolcity $\mathbf{v}_b$ and location $\mathbf{r}_b$. Any quantity A at location $\mathbf{r}_b$ is denoted by A_b. For example, the density can be estimated by equation (2.6) as

$$\rho(\mathbf{r}) = \sum_b m_b W(\mathbf{r} - \mathbf{r}_b, h) \tag{2.7}$$

2.1.2 SPH Approximation of The First Derivative of A Field Function

Now, consider the integral interpolant expression for the gradient of any function $A(\mathbf{r})$:

$$\nabla A_I(\mathbf{r}) = \int \nabla A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h) \, \mathbf{dr}'$$

Using integration by parts and neglecting the part of surface integrals, the above equation becomes

$$\nabla A_I(\mathbf{r}) = \int A(\mathbf{r}') \nabla W(\mathbf{r} - \mathbf{r}', h) \, \mathbf{dr}' \tag{2.8}$$

Therefore, summation approximation gives

$$\nabla A_s(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\mathbf{r} - \mathbf{r}_b, h) \tag{2.9}$$

Equation (2.9) demonstates the fundamental difference of SPH compared with mesh-based numerical methods in approximating the differential operator - the SPH derivative of any function is obtained by differentiating the kernel - rather than by using finite difference, finite element, or finite volume expressions caculated from a grid.

2.1.3 Alternative SPH Approximations of First Derivatives

Based on the integral and summation approximations described in section 2.1.2, SPH formulations of partial differential equations (PDEs) can be derived. In fact, there are numerous ways to derive SPH formulations of PDEs in addition to simply employing equation 2.9. For example, if we write

$$\nabla A = \frac{1}{g}\left(\nabla(gA) - A\nabla g\right) \tag{2.10}$$

where g is a differentiable function. The corresponding summation interpolant, based on (2.9), is

$$\nabla A_a = \frac{1}{g_a} \sum_b m_b \frac{g_b}{\rho_b}\left(A_b - A_a\right) \nabla_a W_{ab} \tag{2.11}$$

where $\nabla_a W_{ab}$ denotes the gradient of $W(\mathbf{r}_a - \mathbf{r}_b, h)$ with respect to the coordinate of the particle a. Note that if we let $g \equiv 1$, (2.11) becomes

$$\nabla A_a = \sum_b \frac{m_b}{\rho_b} (A_b - A_a) \nabla_a W_{ab} \tag{2.12}$$

If we let $g \equiv \rho$, (2.11) becomes

$$\nabla A_a = \frac{1}{\rho_a} \sum_b m_b (A_b - A_a) \nabla_a W_{ab} \tag{2.13}$$

Instead of writing the first derivatives as in (2.10), we can also write

$$\nabla A = g \left[\nabla \left(\frac{A}{g} \right) + \frac{A}{g^2} \nabla g \right] \tag{2.14}$$

Based on (2.9), the corresponding summation interpolant is

$$\nabla A_a = g_a \sum_b \frac{m_b}{\rho_b} g_b \left(\frac{A_b}{g_b^2} + \frac{A_a}{g_a^2} \right) \nabla_a W_{ab} \tag{2.15}$$

Let $g \equiv \rho$, (2.15) becomes,

$$\nabla A_a = \rho_a \sum_b m_b \left(\frac{A_b}{g_b^2} + \frac{A_a}{g_a^2} \right) \nabla_a W_{ab} \tag{2.16}$$

One good propertie of the equations (2.15) - (2.16) , is that quantities A appear in paired particles. Also note that the approximations of (2.11) - (2.13) are exact for A being a constant function, which property is not satisfied by approximations (2.9) and (2.15) - (2.16).

2.2 SPH Discretization of Lagrangian Fluid Equations and Related Algorithms

2.2.1 The Euler equations in Lagrangian form

Based on the laws of conservation of mass, momentum, and energy, the Euler equations in Lagrangian form can be derived. Specifically, based on the conservation of mass, the continuity equation is derived as

$$\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v} \tag{2.17}$$

where $\frac{D}{Dt}$ denotes total time derivative and $\mathbf{v}$ is the velocity field. Similarly, conservation of momentum leads to the momentum equation:

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla P \tag{2.18}$$

where P is the pressure. Finally, the conservation of energy leads to the thermal energy equation:

$$\frac{Du}{Dt} = -\frac{P}{\rho}\nabla \cdot \mathbf{v} \tag{2.19}$$

where u is the thermal energy. We have assumed no viscosity and body forces such as graviational forces. Note that to close the system, an equation of state (EOS) such as $u = f(P, \rho)$, is required.

2.2.2 SPH approximations of Euler equations

There exist numerous forms of SPH approximations of the continuity, momentum, and energy equations, each of which may carry its own advantages and disadvanges. Therefore, the choice of which form to use depends on individual problems. This concept applies to all SPH approximations of PDEs.

The continuity equation can be replaced by approximation of densities. We have seen the most simple density approximation in SPH from equation (2.7), that is, for particle a,

$$\rho_a = \sum_b m_b W_{ab} \tag{2.20}$$

This estimate is usually called *the summation density*, which states that the density of particle a is approximated as the weighted average of the densities of the particles inside the support domain of particle a. However, this approach has its deficiencies when particle a is near boundaries, free surface, or interfaces. For example, since there may not be particles outside the boundaries, the weighted sum gives spurious numerical approximations to the true density. An alternative to estimate density can be derived by using equation (2.13) with the continuity equation (2.17),

$$\frac{D\rho_a}{Dt} = \sum_b m_b \mathbf{v}_{ab} \nabla_a W_{ab} \tag{2.21}$$

where $\mathbf{v}_{ab} = \mathbf{v}_a - \mathbf{v}_b$. With this continuity density estimate, the initial density needs to be set and density will only change when particles move relative to each other. As a result, particles near interfaces will have correct approxima-

tions of densities. One additional advantage of this approach is that rates of change of all physical variables can be computed in one subroutine and there is no need to compute density before other variables, thus saves computational cost and suits computation over parallel processors.

One popular form for the momentum equation can be obtained by employing equation (2.16),

$$\frac{D\mathbf{v}_a}{Dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} \right) \nabla_a W_{ab} \tag{2.22}$$

This approximation is frequently used because it is symmetric in that the force on particle a from particle b is the same as the force on particle b from particle a. Consequently, linear and angular momentum are conserved. Note that artificial viscosity (Π) and external forces such as gravity ($\mathbf{g}$) can be modeled in SPH by rewriting (2.22) as follows (see [14] for details).

$$\frac{D\mathbf{v}_a}{Dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} + \Pi_{ab} \right) \nabla_a W_{ab} + \mathbf{g} \tag{2.23}$$

Using equation (2.13), we can write down the approximation for the equation of energy (2.19):

$$\frac{Du_a}{Dt} = \frac{P_a}{\rho_a^2} \sum_b m_b \mathbf{v}_{ab} \cdot \nabla_a W_{ab}$$

Or if the thermal energy equation (2.19) is rewritten as

$$\frac{Du}{Dt} = -\nabla \left(\frac{P\mathbf{v}}{\rho} \right) + \mathbf{v} \cdot \nabla \left(\frac{P}{\rho} \right)$$

Then the thermal energy equation for particle a can then be approximated using (2.13) as

$$\frac{Du_a}{Dt} = \sum_b m_b \left(\frac{P_b}{\rho_b^2}\right) \mathbf{v}_{ab} \cdot \nabla_a W_{ab}$$

By taking the average of the previous two apporximations, we have

$$\frac{Du_a}{Dt} = \frac{1}{2} \sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2}\right) \mathbf{v}_{ab} \cdot \nabla_a W_{ab} \tag{2.24}$$

which has the same symmetric factors as equation (2.22).

2.2.3 Moving the particles

Particles can be moved according to either

$$\frac{D\mathbf{r}_a}{Dt} = \mathbf{v}_a \tag{2.25}$$

or the XSPH variant ([18])

$$\frac{D\mathbf{r}_a}{Dt} = \mathbf{v}_a + \epsilon \sum_b m_b \left(\frac{\mathbf{v}_{ba}}{\bar{\rho_{ab}}}\right) W_{ab} \tag{2.26}$$

where $\bar{\rho_{ab}} = \frac{\rho_a + \rho_b}{2}$ and ϵ with $0 \leq \epsilon \leq 1$ is a constant. The XSPH variant moves a particle with a velocity that is closer to the average velocity in its neighborhood, which prevents particles from penetrating each other.

2.2.4 Time Integration

Various time integration schemes have been adopted in SPH such as the Predictor-corrector algorithm ([18]), the Verlet algorithm ([35]), and the Symplectic algorithm ([36]). We describe the Predictor-Corrector algorithm here as it is the underlying time-stepping scheme for the SPH simulations presented later in this chapter.

Consider the continuity equation, the equations of momentum and energy, and the equation for mvoing particles in the following form:

$$\begin{aligned}
\frac{D\rho_a}{Dt} &= A_a \\
\frac{D\mathbf{v}_a}{Dt} &= \mathbf{B}_a \\
\frac{Du_a}{Dt} &= C_a \\
\frac{D\mathbf{r}_a}{Dt} &= \mathbf{D}_a
\end{aligned} \tag{2.27}$$

where the calculations are with respect to particle a. The Predictor step gives

$$\begin{aligned}
\rho_a^{n+\frac{1}{2}} &= \rho_a^n + \frac{\Delta t}{2} A_a^n \\
\mathbf{v}_a^{n+\frac{1}{2}} &= \mathbf{v}_a^n + \frac{\Delta t}{2} \mathbf{B}_a^n \\
u_a^{n+\frac{1}{2}} &= u_a^n + \frac{\Delta t}{2} C_a^n \\
\mathbf{r}_a^{n+\frac{1}{2}} &= \mathbf{r}_a^n + \frac{\Delta t}{2} \mathbf{D}_a^n
\end{aligned} \tag{2.28}$$

And the equation of state is used to predict pressure: $P_a^{n+\frac{1}{2}} = f\left(\rho_a^{n+\frac{1}{2}}, u_a^{n+\frac{1}{2}}\right)$. These values are then corrected in the Corrector step using forces at half time

step $(n+\frac{1}{2})$:

$$\begin{aligned}
\rho_a^{n+\frac{1}{2}} &= \rho_a^n + \frac{\Delta t}{2} A_a^{n+\frac{1}{2}} \\
\mathbf{v}_a^{n+\frac{1}{2}} &= \mathbf{v}_a^n + \frac{\Delta t}{2} \mathbf{B}_a^{n+\frac{1}{2}} \\
u_a^{n+\frac{1}{2}} &= u_a^n + \frac{\Delta t}{2} C_a^{n+\frac{1}{2}} \\
\mathbf{r}_a^{n+\frac{1}{2}} &= \mathbf{r}_a^n + \frac{\Delta t}{2} \mathbf{D}_a^{n+\frac{1}{2}}
\end{aligned} \tag{2.29}$$

At the end of the time step, values are computed as

$$\begin{aligned}
\rho_a^{n+1} &= 2\rho_a^{n+\frac{1}{2}} - \rho_a^n \\
\mathbf{v}_a^{n+1} &= 2\mathbf{v}_a^{n+\frac{1}{2}} - \mathbf{v}_a^n \\
u_a^{n+1} &= 2u_a^{n+\frac{1}{2}} - u_a^n \\
\mathbf{r}_a^{n+1} &= 2\mathbf{r}_a^{n+\frac{1}{2}} - \mathbf{r}_a^n
\end{aligned} \tag{2.30}$$

And the pressure is updated by the equation of state: $P_a^{n+1} = f\left(\rho_a^{n+1}, u_a^{n+1}\right)$.

2.2.5 Modelling Solid Boundaries

In SPH the solid boundary is described by a set of discrete boundary particles. These solid boundary particles are modeled to avoid penetration by fluid particles. There are several methods to achieve the expected repulsion, and we review three in this section.

Quasi-fluid boundary particles

[33] proposed to model boundary particles such that their physical properties are evolved to satisfy the same equations as fluid particles during simulation. To be specific, the boundary particles are placed on the boundary and they follow the momentum equation (equation (2.22)), the continuity equation (equations (2.20) or (2.21)), the energy equation (equation (2.24)), and the equation of state. However, these boundary particles remian fixed in position, or move according to certain externally given functions in the case of moving solid boundaries such as wave makers (i.e., they do not move according to equations (2.25) or (2.26)). Since the physical properties of the boundary particles evolve in the same way as the fluid particles but they are either fixed in position or move in a prespecified trajectory, these boundary particles are called "quasi-fluid boundary particles". Consider the equation of state that is used to simulate incompressible fluid such as water in SPH (see [19] for details of this equation of state):

$$P = B\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] \tag{2.31}$$

Consequently, when a fluid particle approaches the boundary the density of the boundary particles increase according to the continuity equation (equations (2.20) or (2.21)), resulting in a pressure increase based on the equation of state (2.31). As a result, the force exerted on the fluid particle increases due to the pressure term $\left(\frac{P}{\rho^2}\right)$ in the momentum equation (equation (2.22)).

Repulsive boundary particles

In [19], solid boundaries are described by boundary particles which exert external forces on fluid particles. Unlike quasi-fluid boundary particles, whose physical properties are updated, states of the repulsive boundary particles remain constant throughout the entire simulation. On the other hand, whether and how the replusive boundary particles contribute to the particle approximation for the fluid particles are problem-dependent.

The form for the force is the Lennard-Jones force. Specifically, for a boundary particle and fluid particle separated by a distance $r = |\mathbf{r}|$ the force per unit mass $\mathbf{f}(\mathbf{r})$ is specified as

$$\mathbf{f}(\mathbf{r}) = \begin{cases} D\left[\left(\frac{r_0}{r}\right)^{p_1} - \left(\frac{r_0}{r}\right)^{p_2}\right]\frac{\mathbf{r}}{r^2} & \text{if } r \leq r_0 \\ 0 & \text{if } r > r_0 \end{cases} \tag{2.32}$$

where the constants p_1 and p_2 must satisfy the condition $p_1 > p_2$, and usually $p_1 = 4$ and $p_2 = 2$ or $p_1 = 12$ and $p_2 = 6$ are specified. The choice of the cutoff distance r_0 is important since if r_0 is too large then some particles may feel the influence of the boudary particles in the initial distribution, and introduce unnecessary initial disturbance. On the other hand, if r_0 is too small, some particles may have penetrated the boundary before they feel the repulsive force. As a result, r_0 is often set to be the initial particle spacing. The coefficient D is problem-dependent (see [19] for details). Note that since $\mathbf{f}(\mathbf{r}) = 0$ when $r > r_0$, the boundary force is purely repulsive. After summing up all the forces of the boundary particles k on the given particle a (i.e., $\mathbf{f}_a = \sum_k \mathbf{f}_{ak}$),

the momentum equation becomes

$$\frac{D\mathbf{v}_a}{Dt} = -\sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} \right) \nabla_a W_{ab} + \mathbf{f}_a \tag{2.33}$$

Mirror boundary particles

[34] introduced mirror particles to reflect a symmetrical surface boundary condition. The mirror particles are constructed in the following way. For a given fluid particle a , if it is located within the distance of κh from the boundary, a mirror particle is then placed symmetrically on the outside of the boundary. For example, for plane boundaries in the Cartesian coordinate, the mirror particles are simply reflected to the other side of the plane (see Figure 2.1 for the case $\kappa = 2$). Nevertheless, for arbitrary shaped boundaries, the mirroring needs to be performed pointwise by considering the local tangent plane.

Similar to repulsive boundary particles, mirror boundary particles *do not* evolve their physical properties. Mirror boundary particles are generated at every time step by reflecting nearby fluid particles (if any) across the solid boundary and their physical properties are deduced from the corresponding fluid particles. There are many ways in doing this. For example, we can assign the density and pressure of the fluid particles directly to the corresponding mirror boundary particles, and negate their velocity. Alternatively, we can decompose the velocity of the fluid particle into normal (u_n) and tangential (u_t) velocity components, and assign to the corresponding mirror boundary

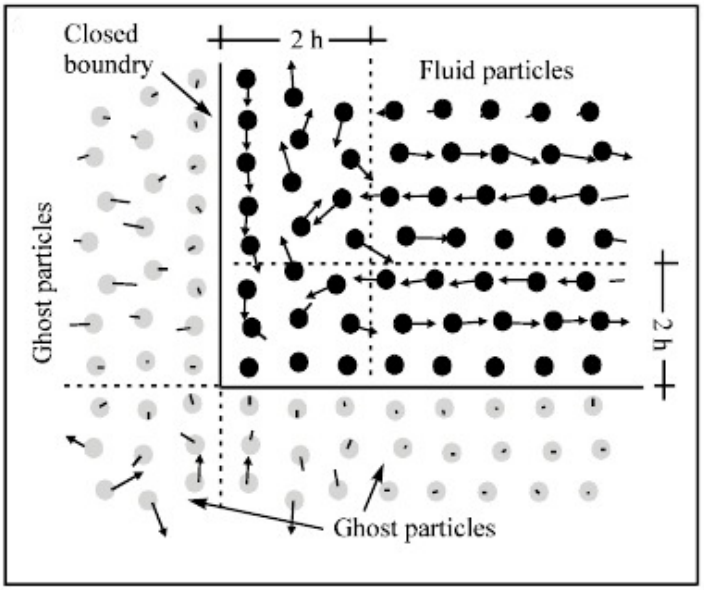


Figure 2.1: 2D configuration of mirror particles. Black dots represent fluid particles while gray dots represent mirror particles. We obtain this figure from [32]: they use the term "Ghost particles" for the mirror particles.

particle as follows:

$$u_{n_{mirror}} = -u_n; \quad u_{t_{mirror}} = u_t$$

.

Choice of Solid Boundary Particles

While there is no universally optimal choice for the modelling of solid boundaries, there are obvious trade-offs among them. For example, mirror boundary particle carries the most accurate physical meaning of "solid boundaries", since they are generated at each time step by reflecting both the location and states of nearby fluid particles (if any). In this sense, they should be able to deliver the most accurate results. However, the generation of mirror boundary parti-

cles is usually a computationally intensive routine and has to be done at each iteration of the simulation. Compared with quasi-fluid boundary particles and repulsive boundary particles, which need to be generated only once during initialization, the choice of using mirror boundary particles will slow down the simulation.

However, both the quasi-fluid boundary particles and the repulsive boundary particles do not carry the most accurate physical meaning in that they are not located at the exact mirror position of nearby fluid particles. While the quasi-fluid boundary particles update states by nearby fluid particles using the SPH equations, they can be viewed as an approximate of mirror boundary particles. As for the repulsive boundary particles, they require aritificial parameters p_1, p_2, γ_0, and D in (2.32), which is ad-hoc and it can be very difficult to find the optimal parameters.

2.2.6 Cavitation Modelling

Hydrodynamic cavitation describes the process of vaporisation, bubble generation and bubble implosion. This occurs in a flowing liquid when the local pressure declines to some point below the saturated vapor pressure of the liquid and subsequent recovery above the vapor pressure.

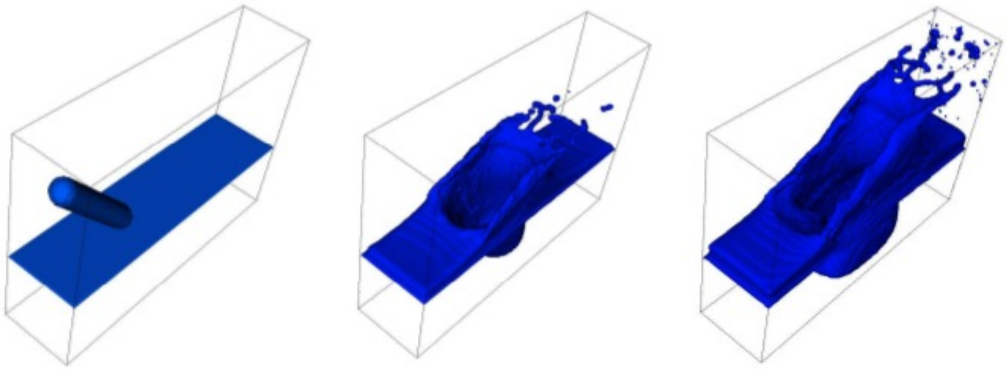
In order to simulate the mercury splash from thimble (see section 2.4.1) and the mercury jet experiments for Neutrino factory and muon collider (see section 2.4.2), the cavitation needs to be modeled. The model we propose for cavitation is based on *critical pressure*. The critical pressure (*cp*) is a pre-

specified constant parameter in the simulation such that if a particle of index i is updated to have pressure p_i and $p_i \leq cp$, then p_i is reset to zero after the update. During the next time step, p_i equals zero, and the nearby particles which take particle i as a neighbour will use this value (zero) in the calculation of the SPH derivatives. As a result, nearby particles which would have been attracted near to partcile i in the case when the original value $p_i < 0$ is used, will not be pulled near to particle i in the case when p_i is reset to 0. In the latter case, those nearby particles may move away from particle i and this helps the formation of cavitation bubbles. We will show the results in sections 2.4.1 and 2.4.2.

2.3 The Parallel SPH Code

The main body of the SPH code for solving the Euler equation was developed by my teammate Tongfei Guo in 2011. The parallelization was also done by Tongfei using the distributed memory parallelization model with MPI. The modules he developed include the following:

1. Controller module
2. Storage module
3. Equation of state module
4. Initialization module
5. SPH Kernel module



(a) Initial state (b) Jet hit the pool (middle state) (c) Jet hit the pool (end state)

Figure 2.2: The simulation of a mercury jet entering into a mercury pool. Plot courtesy Tongfei Guo [51].

6. Solver module

7. Bucket neighbour search and accmulator module

8. Time integration module

9. Parallelization module

Tongfei used the code to run simulations of nearly incompressible fluids, such as the simulation of mercury jet entering into mercury pool. The results are shown in Figure 2.2 (see [51] for the physics background of the simulation).

To run the simulations of the interactions of mecury with proton pulses, I built on Tongfei's code and add the following features to it:

1. The proposed cavitation model (see section 2.2.6)

2. The solid boundary modeled by mirror particles and repulsive particles (see section 2.2.5)

3. The grid-based visualization module

The simulation results of the mercury thimble splash present in section **??** are done with the mirror solid boundary particles and the visulization module will be described in section 2.3.10. In addition to the listed items, during the development of the SPH code, I have also added alternative implementations, such as , different SPH kernels and forms of SPH discretization of the spatial derivatives, to the code. The modules listed above are briefly described in the following sections.

2.3.1 Controller module

The controller controls the work flow of the code. First it reads command line inputs, and calls the initialization module to read input files. The major task of the controller module is to create a loop from the initial time step to the last time step, and call the time integration module at each time step. Inside the loop, the controller calls some pre-and-post time integration routines. For example, before time integration, the lengh of time stepping needs to be calculated (by the CFL condition), and after time integration a check for outputting data for visualization is required.

2.3.2 Storage module

This storage module basically serves as the data storage center for all particle data. Particle data include location and all physical quantities such as pressure, density, velocities, and thermal energy, and sound speed, of parti-

cles. All other quantitities related to particles are also stored in this module, such as the accumulated quantities of the product of some physical quantities of the neighbouring particles and the SPH kernel derivative (which is computed for the temporal derivative of some physical quantities such as density). These data are saved using the primitive data structure - C arrays, to ensure contiguous memory access, giving better efficiency of the code.

2.3.3 Equation of state module

The equation of state (EOS) calculates the pressure and the sound speed based on the information of thermal energy and density. The code implements the polytropic gas EOS and the stiffened polytropic gas EOS. All the simulations done in section 2.4 are based on the stiffened polytropic gas EOS.

2.3.4 Initialization module

This module initializes the goemetry and the physical states of particles. Level set fucntions and routines that generates particles in a cubic or hexagonal packing are implemented for the initialization of particle geoemtry. Particles states are initialized by specifying functions of any two quantities of pressure, density, thermal energy, and using the EOS module to calculate the unspecified one. A complete initialization also includes the assignment of velocities, and reading all other quantities that are specified in the inputfile. Note that all the above functionalities are hard-coded in the module, as a result, it becomes necessary to reqrite code directly in the module. This is error-prone and should

be avoided. The Lagrangian particle code introduced in section 3.3.2 takes care of the initialization based on the design pattern *abstract factory* and prevents this kind of modification, which yields a much safer code.

2.3.5 SPH Kernel module

The kernel module computes the SPH kernels and the SPH kernel derivatives. It is called by the accmulator module with the input q in equation 2.5. There are many kernel functions such as the cubic spline kernel (equation (2.5)). the quadratic and quintic kernel functions. They are all implemented in the code (see [32] for a review on SPH kernels).

2.3.6 Bucket neighbour search and accmulator module

The bucket neighobur search algorithm implements the algorithm introduced in section 3.2.4, which requires only constant time for a particle to search for its neighbours, *if* the fluid is nearly incompressible (i.e. the inter-particle spacing is relatively constant). As a result, for all N particles to search for neighbours, this algorithm runs in $O(N)$ time.

In the compressible case, however, this algorithm fails and must be replaced by other algorithms such as trees. The code only implements the bucket search algorithm and is limited to simulations of weakly compressible fluids.

The accumulator module perfoms the SPH apporximations to the temporal derivatives of physical quantities such as density (eqaution (2.21)) and velocities (equation (2.22)). Different forms of SPH approximation to first

derivatives (see section 2.1.3) are implemented in the module such that combinations can be made in the solver module.

Note that we discuss these two modules together since the accumulator calculates the temporal derivatives at the time when neighbours are searched. This saves space since no neighbour infomation needs to be saved. In the case of constant neighbour search radius, the accumulator adds to both particles when finding neighobours for one particle, due to the fact that each particle takes the other as one of its neighbours. This yields a more efficient code with clearly designed path of nearby buckets in the bucket search algorithm ([17]).

2.3.7 Solver module

This module computes the SPH approximations for the Euler equation as described in section 2.2.2. It solves for the SPH time derivative of pressure, density, and velocities, based on equations (2.21), (2.22), and (2.24), which are are computed by summing up the product of SPH kernel derivatives and some combination of physical quantities of the neighbouring particles. In this module, the summations are done by calling implementations of the accumulator module that represent different SPH approximations to the Euler equation. For instance, the momentum equation can be solved by equation (2.22) or (2.23), which two forms are both implemented in the code.

2.3.8 Time integration module

This module performs time integration for one time step based on algorithms such as predictor-corrector and simpletic schemes. It calls the solver module in between the middle updating steps. The code only implements the predictor-corrector scheme.

2.3.9 Parallelization module

This module parallelizes the code using the distributed-memory model with MPI. First the entire computational domain is divided into sub-domains which is controlled by each distributed cores. Then buffer zones between any of these sub-domains are generated. At the start of each time step, particles lying inside the buffer zones are copied to the neighbouring sub-domains to make each sub-domain *self-sufficient* sub-domains. That is, each core can carry out all required computations during the update of one time step, such as neighbour search and calling the accumulators. As a result, we obtain updated quantities *independently* in each sub-domain. The particles in buffer zone will be re-identified at the end of each time step, and at the begining of the next time step, the updated information in buffer zones will be copied and exchanged again. The exchange of information is done with MPI.

2.3.10 Grid-based visualization module

This module enables the use of contour plots in VisIt by transforming particle-based vtk files to grid-based vtk files. Specifically, the normal output vtk files

of SPH are based on particles, which cannot be visulaized in contour plots, which functionality is provided only for grid-based vtk files. Contour plots are necassary since it resolves interfaces better and enables visualization of quantities inside the fluid objects (by slicing).

This module interpolates particle data to nodes on a regular grid using the SPH kernel functions. That is, for each node in the regular grid it searches nearby particles as neighbours and caculates the weighted average using SPH kernel functions. The results can be seen in Figure 2.9, in which Figure 2.9(a) is based on grid-based visualization and Figure 2.9(b) on particle-based visualization.

2.4 Applications of SPH

In this section we validate SPH simulations of the interaction of mercury with proton pulses. This research supported the high power accelerator target research of the DOE Muon Accelerator Program (MAP).

Liquid metal targets for particle accelerators that convert intense proton beams into neutrons or other particles important for fundamental studies and applied studies. In collider-accelerators or neutrino factories, they are expected to enable new research in high energy / particle physics while in neutron sources technological advances. The aim of the targetry group of MAP is to explore the feasibility of high power targets for future particle accelerators. Our group has performed mathematical modeling, software development and simulations of liquid mercury jet targets interacting with high power pro-

ton beams in magnetic fields. Simulations aimed to make predictions for the first large targetry experiment at CERN called MERIT. The numerical simulations aim to describe the hydrodynamic response of the target interacting with proton pulses in magnetic fields and provide input for the design of reliable targets. Previous simulations used FronTier, a multiphysics code with explicit resolution of material interfaces based on front tracking. As FronTier simulations were successful mostly in lower-power regimes, we used particle-based SPH simulations to extend simulations to high energies and beyond projected experimental conditions. The main conclusion of the targetry program is that liquid mercury jet targets can reliably work in future accelerators and neutron sources up to 8 MW power limit.

We compared the simulation results to the so called mercury thimble experiments at the Brookhaven National Laboratory (BNL) and mercury jet target experiment called MERIT performed at CERN. The reason for the success will be described in the next section (section 2.5).

2.4.1 Mercury Thimble Experiments

In April 2001, Brookhaven National Laboratory (BNL) conducted series of experiments on proton induced shocks using the Alternating Gradient Synchrotron (AGS) beam (at energy 24 GeV) with different target configurations such as a thimble (see [37] for details). The experimental layout of the mercury thimble experiement is presented in Figure 2.3. Specifically, the volume of the thimble excavated in a stainless steel bar is 1.3 cm^3. It consists from

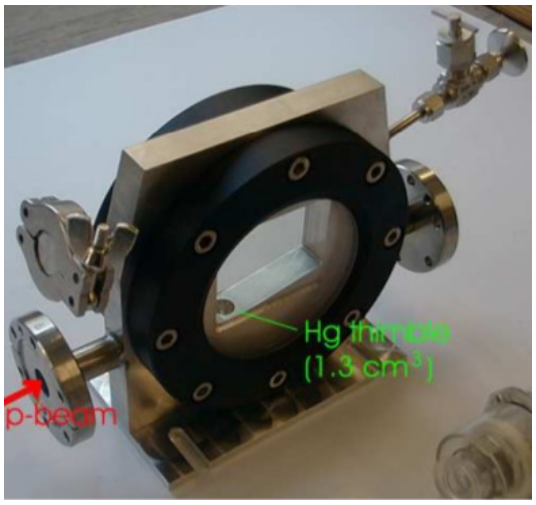


Figure 2.3: The experimental layout of mercury thimble Figure courtesy: [37]
.

bottom to top of a half sphere (r = 6 mm), and a vertical cylinder (r = h = 6 mm). The mercury has a free surface in up-direction, where it can expand to. The proton pulse has approximately Gaussian distribution and the intensity range of $0.6 - 17 \cdot 10^{12}$ protons at energy 24 GeV. The experimental results include a high-speed photography of the liquid targets, which is to record the shadow of the mercury, intercepting a laser light source, with a high speed camera. Figure 2.4 demonstrates the photos of mercury splash resulted from high-intensity proton beam (3.7 10^{12} protons). In addition to qualitatively observe the shapes of mercury splash, the droplet velocities of the mercury splashes resulted from proton-induced shocks with various beam parameters were also recorded. These droplet velocities (free surface behaviours) were observed with various proton beam parameters so that results from desired scenarios (i.e. different beam parameters) can be extrapolated. Figure 2.5 depicts the droplet velocities versus various proton beam intensities.

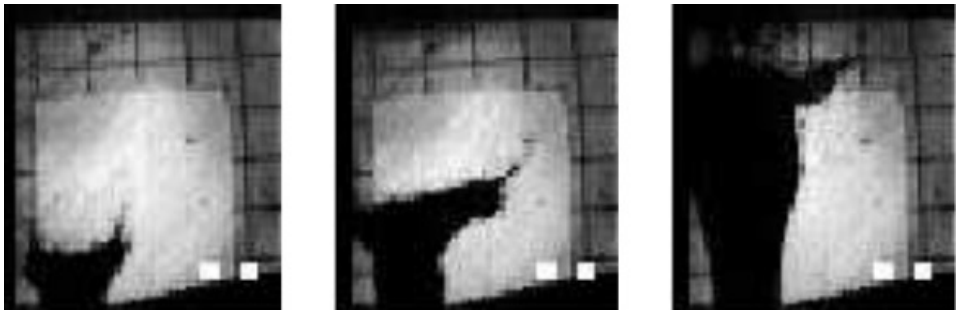


Figure 2.4: 2001 BNL experiment results of the mercury splash from thimble at t = 0.08, 0.125, 0.7 ms after proton impact of 3.7 10^{12} protons, captured by high-speed photography. Figure courtesy: [37]

.

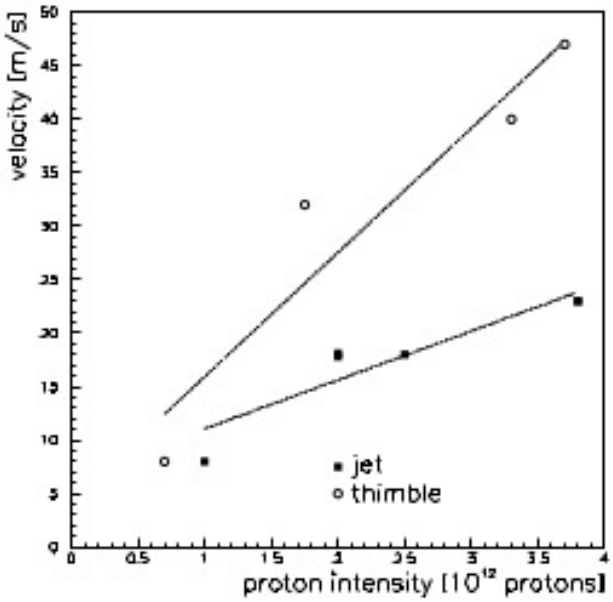


Figure 2.5: Splash velocity of thimble and jet in the AGS beam. The splash velocity in the case of the jet is about two times less than for the thimble. Figure courtesy: [37]

.

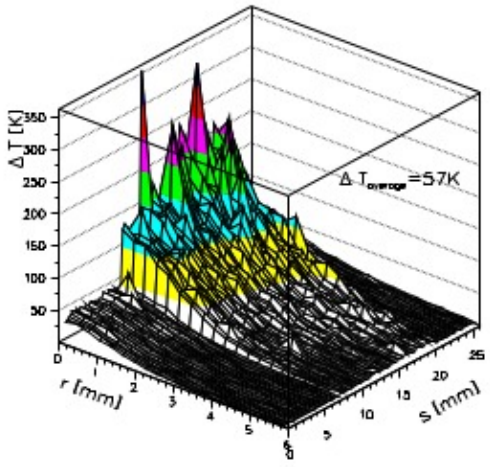
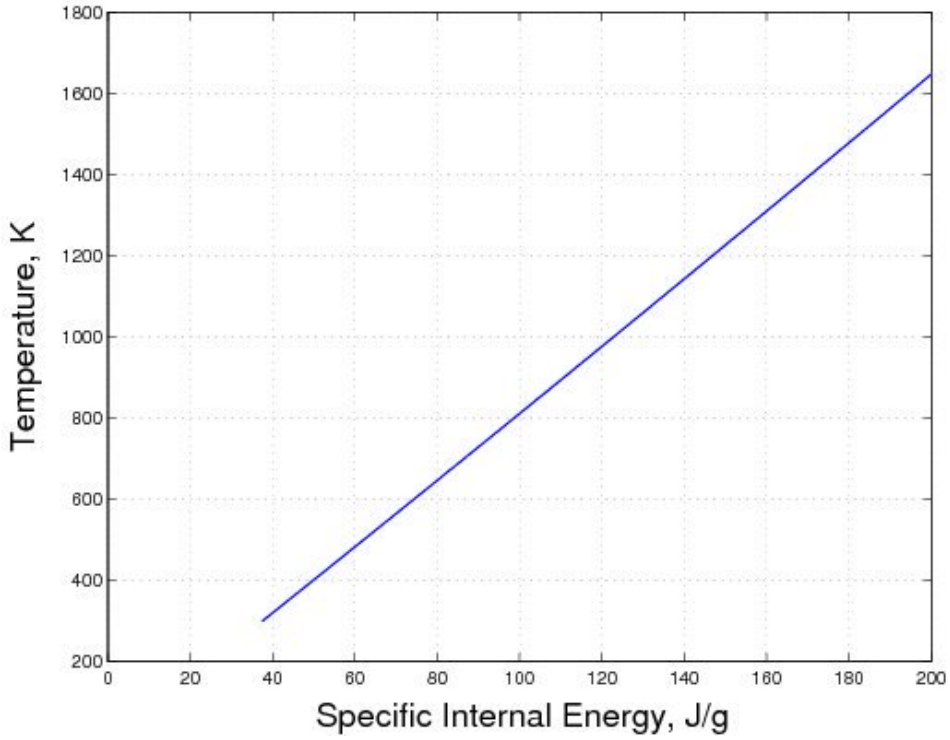


Figure 2.6: Temperature profile for the thimble, 24 GeV, 4 10^{12} protons/pulse. Figure courtesy: [37]
.

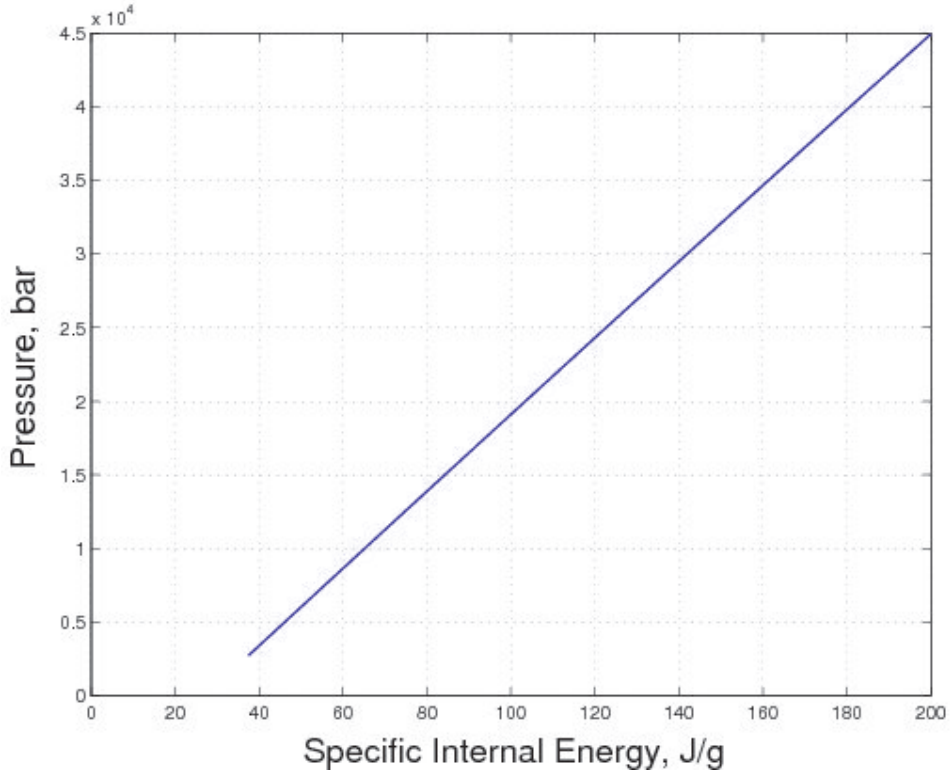
The energy deposition of the thimble under the proton beam shock has been simulated by the code MARS [] and the results has been presented in [37] (see Figure 2.6). Based on the temperature profile shown in Figure 2.6, and the transformations in Figure 2.7, the maximum pressure should be approximately 1.5 10^4 (bar) and we use the following formula to initialize the pressure profile of the numerical simulation.

$$p\text{max} \exp\left[-\left(\frac{(y^2+z^2)}{0.16}+\frac{(x-0.6)^2}{2.7}\right)\right] \tag{2.34}$$

where $p_{\text{max}} = 1.5\ 10^4$ (bar) and the center of the semi-sphere of the thimble is set to be the origin. The geometric configuration and the pressure profile are depiced in Figure 2.8. Other settings include: density is is uniform ($\rho = 13.534$ (g/cm^3)), and the EOS is used is stiffened polytropic gas with $\gamma = 6$, einf $= 0$

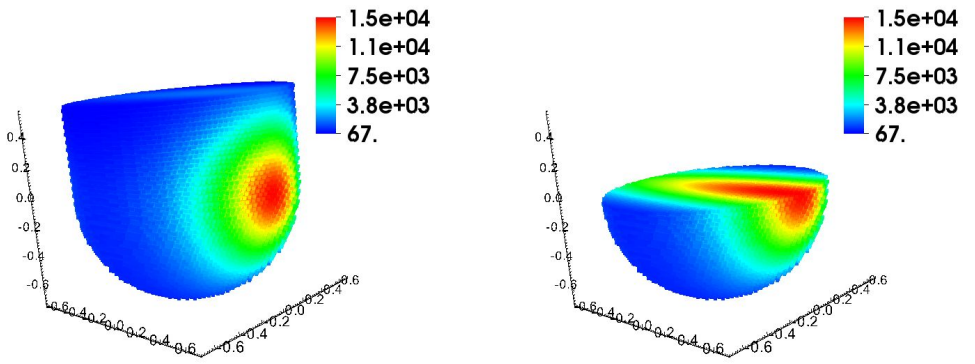


(a)



(b)

Figure 2.7: Isochoric increase of mercury temparature 2.7(a) and pressure 2.7(b) with increase of internal energy (data courtesy Sandia National Lab).

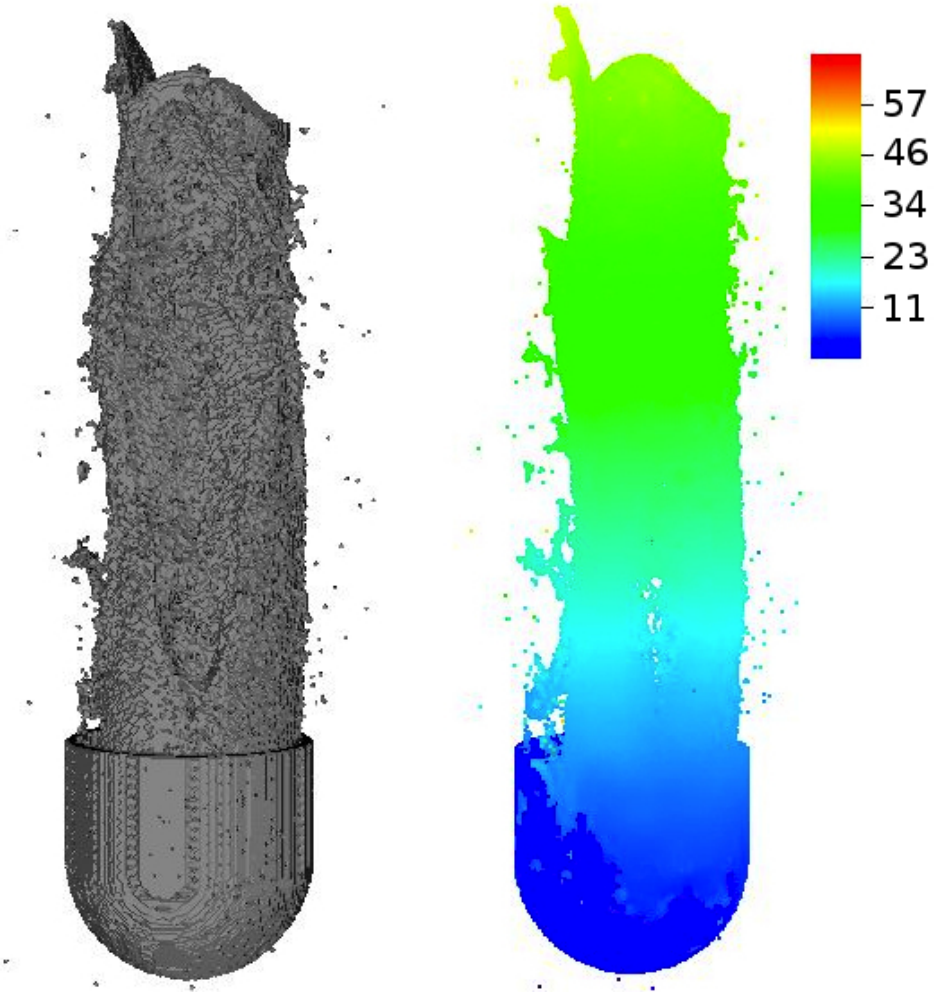


(a) (b)

Figure 2.8: The geometric configuration and pressure profile of the mercury thimble simulation. Figure 2.8(a) plots the thimble with energy deposited at around (0.6,0,0), where is the peak of the Gaussian pressure distribution as specified in (2.34). Figure 2.8(b) plots the pressure profile which only shows the lower part (the semi-sphere part) of the thimble. The upper part of the thimble (the cylinder part) is not shown only for visualization purposes.

and pinf = 47517 (bar).

Base on the above settings we run SPH simulations and show the free surface behaviors in Figure 2.9. Compared with the photos in Figure 2.4, they have similar mushroom shapes on top of the mercury column. Finally, we present the droplet velocities in the z-direction in Figure 2.10. Note that in Figure 2.5 droplet velocities under the intensity of 3.7 10^{12} is about 45 m/s. This corresponds to the simulated droplet velocities (see Figure 2.10) under the condition that $p_{\max} = 15$ kbar, which is around 42 m/s. From the simulation results of the thimble mercury splash, we observed that the splash velocity is sensitive to the value of the *critical pressure* used in the cavitation model, as described in section 2.2.6. Good agreement with experimental data



(a) Marcury surface (b) z-velocity, m/s

Figure 2.9: Simulation of the thimble experiment. Shape (left) and vertical velocity (right) of the mercury splash column after 0.85 ms.

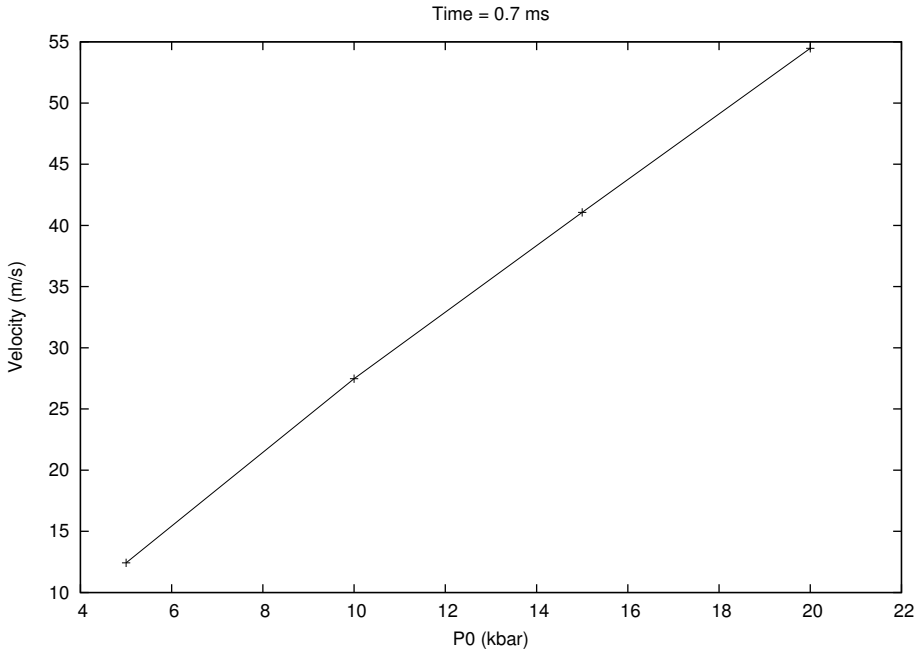


Figure 2.10: Splash velocity of thimble observed from simulation results with $p_{\max}$ equal to 10, 15, and 20 kbars.

was obtained for the critical pressure of 100 bar. This value was used in the simulations shown in this section and all mercury jet simulations in the follwing section.

2.4.2 Mercury Targets for Neutrino Factory and Muon Collider

In this section, we report simulation results of the mercury jet interaction with proton pulses in the neutrino factory and muon collider power regimes. The 4 MW beam of 8 GeV protons will be delivered in 150 bunches per second for the neutrino factory and in 15 bunches per second for the muon collider. 20.8 teraproton bunches arrive at the neutrino factory target with the 6.67 ms

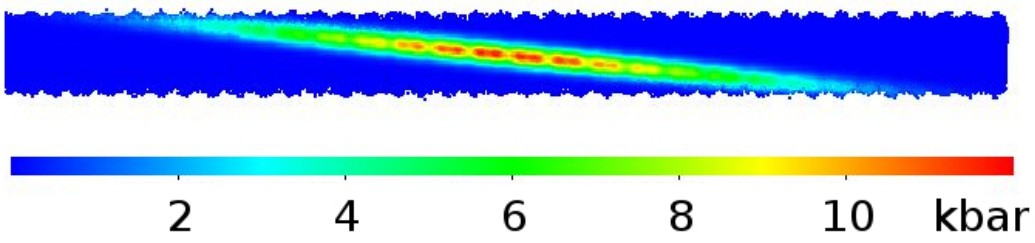


Figure 2.11: Initial pressure distribution in cross-section of the mercury jet due to neutrino factory beam energy deposition.

interval that is large compared to hydrodynamic time scales of the mercury splash. In the case of the muon collider, 208 teraproton bunches arrive with the time interval of 66.7 ms. Therefore it is sufficient to consider the interaction of the mercury jet with only one proton pulse. We use the corresponding energy deposition tables calculated at FNAL with the Monte Carlo code MARS (see Figure 2.7). Assuming the isochoric regime of the energy deposition (the deposition time scale is mush smaller than the hydrodynamic time scale), the peak pressure is approximately 110 kbar in the muon collider target and 11 kbar in the target for the neutrino factory. The proton pulse enters the jet under small angle and the energy deposition is not axially symmetric. The energy deposition profile in the initial jet is shown in Figure 2.11.

After the energy deposition, strong pressure wave propagates to the jet surface and reflects from the mercury - vacuum interface as a rarefaction wave. Rarefaction waves focus in the center of the jet, break the liquid and create an extensive cavitation zone. After the jet is disintegrated by cavitation and surface instabilities, fragments of jet freely fly apart. The velocity of the neutrino factory target splash is in the range of 15 - 35 m/s, with some small

droplets reaching velocities of the order of 40 - 50 m/s (see Figures 2.12(a) and 2.12(b)). These velocities are in the same range as ones observed in MERIT experiments. The exact comparison is not possible because of uncertainties in the energy deposition and spot sizes. In the muon collider case, main jet fragments disperse with the velocity of 90 - 110 m/s with some droplets reaching much higher velocities (see Figure 2.12(c)). The simulation results using 24 GeV, 12 teraprotons proton pulse is shown in Figure 2.13.

2.5 Limitations of SPH

2.5.1 Analysis of Accuracy

Approximation of field function

Assume that the function A is sufficiently smooth, we can apply Taylor series expansion of $A(\mathbf{r}^{'})$ in the vicinity of $\mathbf{r}$,

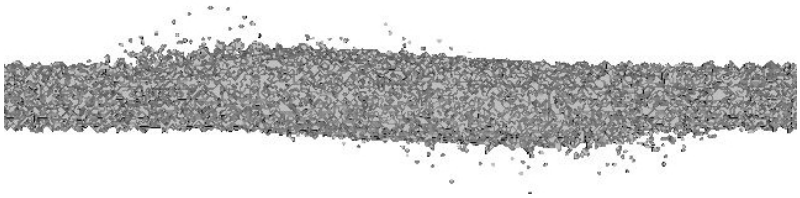
$$A(\mathbf{r}^{'}) = A(\mathbf{r}) + A^{'}(\mathbf{r})(\mathbf{r}^{'} - \mathbf{r}) + \frac{1}{2}A^{''}(\mathbf{r})(\mathbf{r}^{'} - \mathbf{r})^2 + \cdots \tag{2.35}$$

Substitute (2.35) into (2.2), and assume that

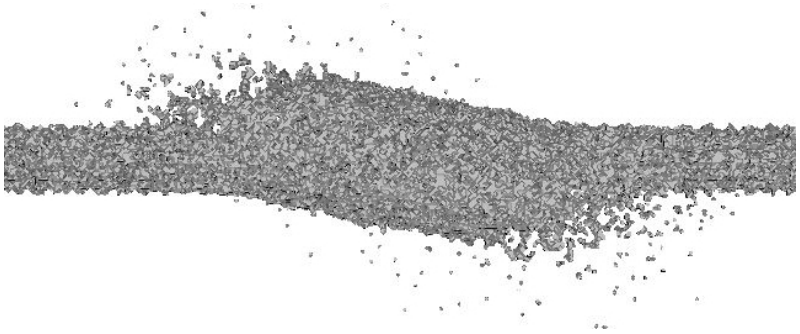
$$\int_{\Omega} W(\mathbf{r}^{'} - \mathbf{r}, h)d\mathbf{r}^{'} = 1 \text{ (unity equation)} \tag{2.36}$$

and

$$\int_{\Omega} (\mathbf{r}^{'} - \mathbf{r})W(\mathbf{r}^{'} - \mathbf{r}, h)d\mathbf{r}^{'} = 0 \text{ (symmetric kernel)} \tag{2.37}$$



(a) Neutrino factory, 0.35 ms



(b) Neutrino factory, 1.0 ms

(c) Muon collider, 0.35 ms

Figure 2.12: Mercury jet dispersal after the deposition of 4 MW beam of 8 GeV protons. State of the neutrino factory jet target at 0.35 ms (a) and 1 ms (b), and the muon collider target at 0.35 ms (c).

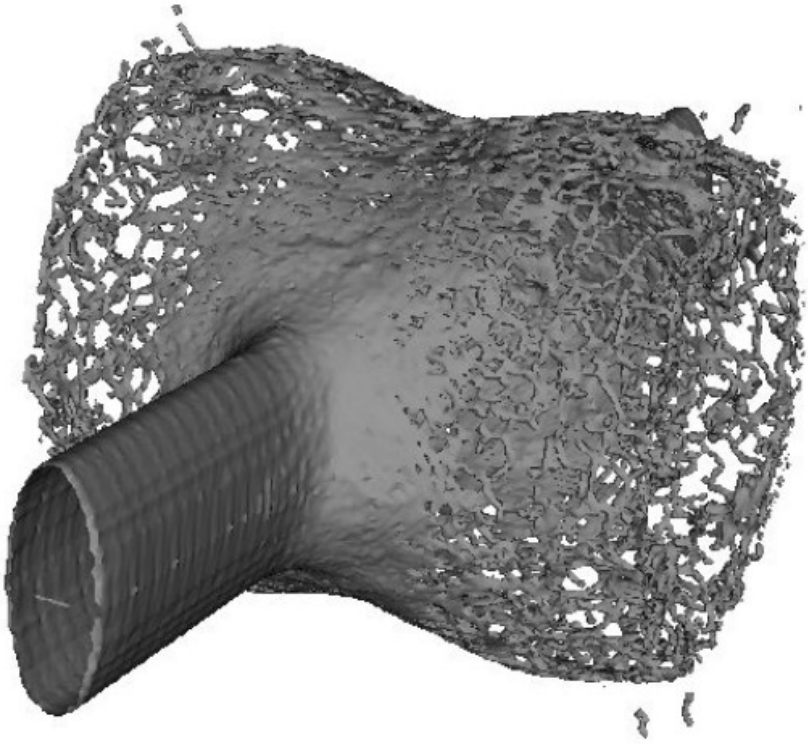


Figure 2.13: Mercury target after interaction with 24 Gev, 12 teraproton proton pulse.

Then the error introduced by (2.2) is

$$A(\mathbf{r}) = \int A(\mathbf{r}')\, W(\mathbf{r} - \mathbf{r}', h)\, \mathbf{dr}' + O\left(h^2\right) \tag{2.38}$$

unless higher order kernels are used. The error of the discrete approximation (2.6) for (2.2), however, depends on how well the summations approximate the integral. Following [22], we expand A_b by Taylor series expansion around $\mathbf{r}_a$ in (2.6), which yields

$$\begin{aligned} A_a &= \sum_b m_b \frac{A_b}{\rho_b} W_{ab} \\ &= A_a \sum_b \frac{m_b}{\rho_b} W_{ab} + \nabla A_a \sum_b \frac{m_b}{\rho_b} (\mathbf{r}_b - \mathbf{r}_a) W_{ab} + O(h^2) \end{aligned} \tag{2.39}$$

This shows that the SPH approximation of the field function A is truly second order accurate if the following conditions hold:

$$\sum_b \frac{m_b}{\rho_b} W_{ab} \approx 1 \tag{2.40}$$

and

$$\sum_b \frac{m_b}{\rho_b} (\mathbf{r}_b - \mathbf{r}_a) W_{ab} \approx 0 \tag{2.41}$$

[22] argues that (2.41) is easier to satisfy than (2.40), since it requires only a reasonably symmetric particle distribution. (2.40), however, depends strongly on the particle distribution within kernel radius and the ratio of smoothing radius to the particle spacing (Δp). In general, the above two conditions are satisfied if the particle distribution is *regular*. To summarize, we quote

from [22]:

> *Thus, maintaining a regular particle arrangement, together with an appropriate choice of* $\frac{h}{\Delta p}$ *, can be very important in obtaining accurate results in SPH.*

Approximation of First Derivatives

Based on the theorem of integration by parts, we have

$$\nabla A(\mathbf{r}) = \int_S A(\mathbf{r}')W\left(\mathbf{r}' - \mathbf{r}, h\right) \cdot \mathbf{n} dS - \int A(\mathbf{r}')\nabla W(\mathbf{r}' - \mathbf{r}, h)\mathbf{dr}' \tag{2.42}$$

Assume even kernel, that is,

$$\int_S f(\mathbf{r}')W\left(\mathbf{r}' - \mathbf{r}, h\right) \cdot \mathbf{n} dS = 0 \tag{2.43}$$

for any function f, Then apply (2.35) to (2.42), we have

$$\begin{aligned} \nabla A(\mathbf{r}) = A(\mathbf{r}) \int \nabla W(\mathbf{r} - \mathbf{r}', h)\mathbf{dr}' \\ + \nabla A(\mathbf{r}) \int (\mathbf{r}' - \mathbf{r})\nabla W(\mathbf{r} - \mathbf{r}', h)\mathbf{dr}' + O(h^2) \end{aligned} \tag{2.44}$$

Since even kernel is assumed,

$$\begin{aligned} \int \nabla W \mathbf{dr}' &= \int_S 1 \cdot W \cdot \mathbf{n} dS - \int 1' \cdot W \mathbf{dr}' \\ &= 0 - 0 = 0 \end{aligned} \tag{2.45}$$

By integration by parts,

$$\begin{aligned}\int(\mathbf{r}'-\mathbf{r})^k W(\mathbf{r}'-\mathbf{r},h)d\mathbf{r}' &= \frac{1}{k+1}\int\left[(\mathbf{r}'-\mathbf{r})^{k+1}\right]' W(\mathbf{r}'-\mathbf{r},h)d\mathbf{r}' \\ &= \frac{1}{k+1}[\int_S(\mathbf{r}'-\mathbf{r})^{k+1}W(\mathbf{r}'-\mathbf{r},h)\cdot\mathbf{n}dS \\ &- \int(\mathbf{r}'-\mathbf{r})^{k+1}W'(\mathbf{r}'-\mathbf{r},h)d\mathbf{r}'] \\ &= -\frac{1}{k+1}\int(\mathbf{r}'-\mathbf{r})^{k+1}W'(\mathbf{r}'-\mathbf{r},h)d\mathbf{r}' \qquad (2.46)\end{aligned}$$

Then if we assume that (2.36) is satisfied, we have $\int(\mathbf{r}'-\mathbf{r})\nabla W(\mathbf{r}-\mathbf{r}',h)\mathbf{dr}' = 1$. The resulting error of equation (2.44) is therefore truly second order if (2.36) and (2.45) hold.

Similar to the previous subsection, the errors of equation (2.9) can be found by expanding A_b in a Taylor series around $\mathbf{r}_a$. That is,

$$\begin{aligned}\nabla A_s(\mathbf{r}_a) &= \sum_b m_b\frac{A_b}{\rho_b}\nabla_a W_{ab} \\ &= A_a\sum_b\frac{m_b}{\rho_b}\nabla_a W_{ab} + \nabla_a A\sum_b\frac{m_b}{\rho_b}(\mathbf{r}_b-\mathbf{r}_a)\nabla_a W_{ab} + O(h^2) \qquad (2.47)\end{aligned}$$

Note that the summations in (2.47) approximate the integrals in (2.44). As a result, the errors resulted from (2.9) depends on how well these integrals are approximated by these summations. That is, how well the following conditions hold.

$$\sum_b\frac{m_b}{\rho_b}\nabla_a W_{ab}\approx 0 \qquad (2.48)$$

and

$$\sum_b \frac{m_b}{\rho_b}(\mathbf{r}_b - \mathbf{r}_a)\nabla_a W_{ab} \approx 1 \tag{2.49}$$

Discussion on SPH First Derivatives

Note that we can have exact SPH first derivative for constant functions if we subtract the first term in (2.47). That is,

$$\nabla A_s\left(\mathbf{r}_a\right) = \sum_b m_b \frac{(A_b - A_a)}{\rho_b}\nabla_a W_{ab} \tag{2.50}$$

We can even have the SPH first derivative for linear functions by solving

$$\chi \nabla_a A = \sum_b m_b \frac{(A_b - A_a)}{\rho_b}\nabla_a W_{ab} \tag{2.51}$$

where

$$\chi \equiv \sum_b \frac{m_b}{\rho_b}(\mathbf{r}_b - \mathbf{r}_a)\nabla_a W_{ab} \tag{2.52}$$

Nevertheless, none of the above forms of SPH derivatives correspond to that which conserves momentum (see [22] for details) is:

$$\nabla A_a = \rho_a \sum_b m_b \left(\frac{A_b}{\rho_b^2} + \frac{A_a}{\rho_a^2}\right)\nabla_a W_{ab} \tag{2.53}$$

Note that (2.53) is the same as (2.16) with g replaced with ρ. Expanding A_b in a Taylor series around $\mathbf{r}_a$ in (2.53), we have

$$\rho_a A_a \sum_b m_b \left(\frac{1}{\rho_b^2} + \frac{1}{\rho_a^2}\right)\nabla_a W_{ab} + \nabla A_a \rho_a \sum_b \frac{m_b}{\rho_b^2}(\mathbf{r}_b - \mathbf{r}_a)\nabla_a W_{ab} + O(h^2) \tag{2.54}$$

Although we could substract the first term in (2.54) to make the approximation exact for constant functions, it will no longer conserve momentum. However, with more less "accurate" SPH derivatives such as (2.53), local conservation of momentum between particle pairs is reserved and thus parpticle distribution will remain more "regular" during simulation. This is why for non-linear problems such as those that run for a long time or involve strong shocks, conservation of momentum is more important than linear errors - since the conservation of momentum results in more stability and the corresponding derivatives may be more accurate than those with less linear errors due to a more regular particle distribution.

Relation of SPH to Hamiltonian Dynamics of Particles

The reason why SPH produces stable and reasonable results for certain problems, despite using inaccurate and non-convergent discretization of differential operators, is its connection to the Lagrangian / Hamiltonian dynamics of particles [22]. In particular, the traditional discrete SPH equations for the compressible Euler equations are not accurate, but they accurately represent equations of the Lagrangian dynamics of particles interacting via isentropic potentials. The Lagrangian and Hamiltonian properties are also responsible for the long term stability of the traditional SPH. But the Hamiltonian dynamics of particles only approximately represent the dynamics of continuum hydrodynamic systems, and the isentropic interaction energy places additional restrictions.

Chapter 3

A New Lagrangian Particle Method

3.1 Governing Equations

Consider the one-dimensional Lagrangian formulation of the Euler equations, written in the conservative form [6, 7]

$$U_t^{'} + \Big[F(U^{'})\Big]_x = 0, \tag{3.1}$$

$$U^{'} = \begin{pmatrix} V \\ u \\ E \end{pmatrix}, \; F(U^{'}) = V \begin{pmatrix} -u \\ P \\ Pu \end{pmatrix}, \tag{3.2}$$

where V is the specific volume, u is the velocity, E is the specific total energy, and P is the pressure.

Assume that the equation of state (EOS) is of the form $e = f(P, V)$, where e is the specific internal energy, $e = E - u^2/2$. Equations (3.1) and (3.2) can be written using $U = [V\ u\ P]^T$ as the state vector

$$U_t + A(U)U_x = 0, \tag{3.3}$$

$$U = \begin{pmatrix} V \\ u \\ P \end{pmatrix}, \quad A(U) = V \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & K & 0 \end{pmatrix}, \tag{3.4}$$

where

$$K = \left(P + \frac{\partial e}{\partial V} \right) \Big/ \frac{\partial e}{\partial P}\,. \tag{3.5}$$

For example, using the polytropic gas EOS

$$e = \frac{PV}{\gamma - 1}, \tag{3.6}$$

where γ is the ratio of specific heats, we obtain

$$A(U) = V \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & (\frac{c}{V})^2 & 0 \end{pmatrix}, \tag{3.7}$$

where $c = \sqrt{\gamma PV}$ is the speed of sound. Using the stiffened polytropic gas EOS

$$e = \frac{(P + \gamma P_{\text{inf}})}{\gamma - 1} - e_{\text{inf}}, \tag{3.8}$$

where the sound speed is calculated by

$$c = \sqrt{\gamma(P + P_{\text{inf}})V} \tag{3.9}$$

K is the same as in the case of polytropic gas EOS (i.e. $k = (\frac{c}{V})^2$). Note that the transformation is exact (i.e. not a result of linerization). If the matrix A is diagonalized as $A = R\Lambda R^{-1}$, equations (3.3) and (3.4) become

$$U_t + R\Lambda R^{-1} U_x = 0,$$

$$R^{-1}U_t + \Lambda R^{-1} U_x = 0, \tag{3.10}$$

where

$$R^{-1} = \begin{pmatrix} 1 & 0 & \frac{1}{K} \\ 0 & -\frac{1}{2\sqrt{K}} & -\frac{1}{2K} \\ 0 & \frac{1}{2\sqrt{K}} & -\frac{1}{2K} \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -\sqrt{K} & \sqrt{K} \\ 0 & -K & -K \end{pmatrix},$$

$$\Lambda = V \begin{pmatrix} 0 & & \\ & \sqrt{K} & \\ & & -\sqrt{K} \end{pmatrix} \tag{3.11}$$

Based on the governing equations (3.10) and (3.11), we have developed stable, particle-based, upwinding numerical schemes for the system of Euler's equations. Details are described in the next section.

3.2 Numerical Discretization and Main Algorithms

3.2.1 Discrete Lagrangian Equtions

To solve numerically the hyperbolic system of PDE's (3.10) and (3.11), the medium (compressible fluid or gas) is discretized by a distribution of particles. Each particle represents a Lagrangian fluid cell of equal mass, and stores states of the continuum medium such as the density (that is proportional to the number density of Lagrangian particles), pressure, internal energy, velocity, as well as material properties and pointers to data structures containing material models, such as the EOS.

To construct a Lagrangian upwinding scheme, we represent the system (3.10)-(3.11) in the following component-wise form

$$
\begin{aligned}
V_t + \frac{1}{K} P_t &= 0, && (3.12)\\
-\frac{1}{2\sqrt{K}} u_t - \frac{1}{2K} P_t &= -V\sqrt{K}\left[-\frac{1}{2\sqrt{K}} u_x - \frac{1}{2K} P_x\right], && (3.13)\\
\frac{1}{2\sqrt{K}} u_t - \frac{1}{2K} P_t &= V\sqrt{K}\left[\frac{1}{2\sqrt{K}} u_x - \frac{1}{2K} P_x\right]. && (3.14)
\end{aligned}
$$

As $K > 0$ for a thermodynamically consistent EOS, equation (3.13) describes waves propagating from left to right, and equation (3.14) describes waves propagating from right to left. For an upwinding scheme, the spatial derivatives u_x and P_x will be computed on stencils within the corresponding physical domains of dependence. Adding the subscripts l and r to the spatial derivatives

in equations (3.13) and (3.14), respectively, to indicate that these terms, in the discrete form, will be computed using one-sided derivatives, and solving for the temporal derivatives, we obtain

$$V_t = \frac{V}{2}\left(u_{xr} + u_{xl}\right) - \frac{V}{2\sqrt{K}}\left(P_{xr} - P_{xl}\right), \tag{3.15}$$

$$u_t = \frac{V\sqrt{K}}{2}\left(u_{xr} - u_{xl}\right) - \frac{V}{2}\left(P_{xr} + P_{xl}\right), \tag{3.16}$$

$$P_t = -\frac{VK}{2}\left(u_{xr} + u_{xl}\right) + \frac{V\sqrt{K}}{2}\left(P_{xr} - P_{xl}\right). \tag{3.17}$$

Note that these one-sided derivatives P_{xr}, P_{xl}, u_{xr}, and u_{xl}, are computed based on neighbouring particles which are to-the-right or to-the-left of the particle of interest. For example, to compute P_{xr} and u_{xr}, only the neighbouring particles to-the-right of the particle of interest are utilized. This concept easily generalizes to higher dimensions.

An important component of a particle-based numerical scheme is the calculation of differential operators based on states at the location of particles. In Section 3.3, we describe in detail a method for both numerical differentiation and interpolation based on local polynomial fitting. In this section, we simply assume that we can compute numerical approximations of differential operators with a desired degree of accuracy on particle-based stencils located in the physical domains of dependance.

The first-order ($O(\Delta t, \Delta x)$ upwinding discretization of the system (3.15-3.16) is obtained by the first-order discretization of spatial derivatives based on the local polynomial fitting, and the first-order discretization of temporal

derivatives of the state (V, u or P) at the location of particle j,

$$\frac{\text{state}_j^{n+1} - \text{state}_j^n}{\Delta t}.$$

After the updates of states of each Lagrangian particle, particles are advanced by a mixture of the forward Euler scheme and backward Euler scheme:

$$\frac{x^{n+1} - x^n}{\Delta t} = \frac{1}{2}\left(u^n + u^{n+1}\right) \tag{3.18}$$

The first order schemes is stable, provided that the standard CFL condition is satisfied: $dt <= l/max(c, u)$, where l is the smallest interparticle distance. To reduce the amount of numerical diffusion of the first-order scheme and to obtain a higher order scheme, we propose a *modified Beam-Warming* scheme for the Lagrangian particle system. For the same reason as in the original work on the Beam-Warming method [23], an additional term is added to equation (3.3):

$$\begin{aligned} U_t + A(U)U_x - \frac{\Delta t}{2}A^2(U)U_{xx} &= 0 \\ \Rightarrow U_t = -R\Lambda R^{-1}U_x + \frac{\Delta t}{2}R\Lambda^2 R^{-1}U_{xx} & \\ \Rightarrow R^{-1}U_t = -\Lambda R^{-1}U_x + \frac{\Delta t}{2}\Lambda^2 R^{-1}U_{xx} & \end{aligned} \tag{3.19}$$

Equations (3.13) and (3.14) then become

$$\begin{aligned} -\frac{1}{2\sqrt{K}}u_t - \frac{1}{2K}P_t &= -V\sqrt{K}\left[-\frac{1}{2\sqrt{K}}u_{xl} - \frac{1}{2K}P_{xl}\right] \\ &+ \frac{\Delta t}{2}V^2K\left[-\frac{1}{2\sqrt{K}}u_{xxl} - \frac{1}{2K}P_{xxl}\right] \qquad (3.20) \\ \frac{1}{2\sqrt{K}}u_t - \frac{1}{2K}P_t &= V\sqrt{K}\left[\frac{1}{2\sqrt{K}}u_{xr} - \frac{1}{2K}P_{xr}\right] \\ &+ \frac{\Delta t}{2}V^2K\left[\frac{1}{2\sqrt{K}}u_{xxr} - \frac{1}{2K}P_{xxr}\right] \qquad (3.21) \end{aligned}$$

Solving equations (3.12), (3.20) and (3.21) yields

$$\begin{aligned} V_t &= \frac{V}{2}\left(u_{xr} + u_{xl}\right) - \frac{V}{2\sqrt{K}}\left(P_{xr} - P_{xl}\right) \\ &+ \frac{\Delta t}{4}\left[V^2\sqrt{K}\left(u_{xxr} - u_{xxl}\right) - V^2\left(P_{xxr} + P_{xxl}\right)\right] \qquad (3.22) \\ u_t &= \frac{V\sqrt{K}}{2}\left(u_{xr} - u_{xl}\right) - \frac{V}{2}\left(P_{xr} + P_{xl}\right) \\ &+ \frac{\Delta t}{4}\left[V^2K\left(u_{xxr} + u_{xxl}\right) - V^2\sqrt{K}\left(P_{xxr} - P_{xxl}\right)\right] \qquad (3.23) \\ P_t &= -\frac{VK}{2}\left(u_{xr} + u_{xl}\right) + \frac{V\sqrt{K}}{2}\left(P_{xr} - P_{xl}\right) \\ &+ \frac{\Delta t}{4}\left[-V^2K^{\frac{3}{2}}\left(u_{xxr} - u_{xxl}\right) + V^2K\left(P_{xxr} + P_{xxl}\right)\right] \qquad (3.24) \end{aligned}$$

Note that in both equations (3.15) and (3.22), V_t can be calculated by the relation: $V_t = -\frac{P_t}{K}$. By discretizing spatial derivatives using the second order local polynomial fitting, as described in Section 3.3, we obtaine numerical scheme that is second order in both time and space, $O(\Delta t^2, \Delta x^2, \Delta t \Delta x)$, and conditionally stable. The CFL condition is simlar to the one of the grid-based Beam-Warming scheme: in 1D, $dt <= 2l/max(c, u)$. Note that time steps can

be twice larger compared to the first-order scheme.

3.2.2 Time Integration and Directional Splitting

In this section, we focus on details of multidimensional schemes. We present explicit formulas for equations in the three-dimensional space. The system in the two-dimensional space is obtained by obvious reductions.

In the three-dimensional space, the conservative form of the Lagrangian formulation of the Euler equations is:

$$U_t^{'} + \Big[F_1(U^{'})\Big]_x + \Big[F_2(U^{'})\Big]_y + \Big[F_3(U^{'})\Big]_z = 0, \tag{3.25}$$

where

$$U^{'} = \Big[\begin{array}{ccccc} V & u & v & w & E \end{array}\Big]^T,$$

$$F_1(U^{'}) = V \begin{pmatrix} -u \\ P \\ 0 \\ 0 \\ Pu \end{pmatrix}, \; F_2(U^{'}) = V \begin{pmatrix} -v \\ 0 \\ P \\ 0 \\ Pv \end{pmatrix}, \; F_3(U^{'}) = V \begin{pmatrix} -w \\ 0 \\ 0 \\ P \\ Pw \end{pmatrix}. \tag{3.26}$$

Assuming that the EOS is of the form $e = f\,(P, V)$ and using $U = [V \; u \; v \; w \; P]^T$ as the state vector, we can rewrite the equations in the following form

$$U_t + A_1 U_x + A_2 U_y + A_3 U_z = 0, \tag{3.27}$$

where

$$U = \begin{pmatrix} V \\ u \\ v \\ w \\ P \end{pmatrix}, \quad A_1 = V \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & K & 0 & 0 & 0 \end{pmatrix},$$

$$A_2 = V \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & K & 0 & 0 \end{pmatrix}, \quad A_3 = V \begin{pmatrix} 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & K & 0 \end{pmatrix}, \tag{3.28}$$

where K is defined in equation (3.5). We solve the system of hyperbolic PDEs (3.27 - 3.28) by using the directional splitting method by Strang [24]. Specifically, instead of solving equation (3.27), one solves separately the following three system of PDEs:

$$U_t + 3A_1 U_x = 0, \tag{3.29}$$

$$U_t + 3A_2 U_y = 0, \tag{3.30}$$

$$U_t + 3A_3 U_z = 0, \tag{3.31}$$

which is equivalent to solving

$$U_{1t} + 3AU_{1x} = 0, \tag{3.32}$$

$$U_{2t} + 3AU_{2y} = 0, \tag{3.33}$$

$$U_{3t} + 3AU_{3z} = 0, \tag{3.34}$$

where

$$U_1 = \begin{pmatrix} V \\ u \\ P \end{pmatrix}, U_2 = \begin{pmatrix} V \\ v \\ P \end{pmatrix}, U_3 = \begin{pmatrix} V \\ w \\ P \end{pmatrix}, A = V \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & K & 0 \end{pmatrix}. \tag{3.35}$$

Each of the three system of equations (3.32) - (3.34) is solved by the techniques introduced in section 3.2.1, and the solutions are combined in the following order

$$\begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(0)} \xrightarrow{(3.32)} \begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(\frac{\Delta t}{6})} \xrightarrow{(3.33)} \begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(\frac{2\Delta t}{6})} \xrightarrow{(3.34)} \begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(\frac{4\Delta t}{6})}$$

$$\xrightarrow{(3.33)} \begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(\frac{5\Delta t}{6})} \xrightarrow{(3.32)} \begin{pmatrix} V \\ u \\ v \\ w \\ p \end{pmatrix}^{(\Delta t)} \tag{3.36}$$

where Δt denotes one discrete time step satisfying the CFL condition. The Strang splitting method maintains the second order of accuracy if the accuracy of each step is not lower than second, making it unnecessary for the first order numerical scheme. To implement the modified Beam-Warming scheme within the Strang splitting steps (3.32) - (3.34), we solve the following equations

$$U_{1t} + 3\left(AU_{1x} - \frac{\Delta t}{2}A^2U_{1xx}\right) = 0, \tag{3.37}$$

$$U_{2t} + 3\left(AU_{2y} - \frac{\Delta t}{2}A^2U_{2yy}\right) = 0, \tag{3.38}$$

$$U_{3t} + 3\left(AU_{3z} - \frac{\Delta t}{2}A^2U_{3zz}\right) = 0. \tag{3.39}$$

The solutions to equations (3.37) - (3.39) are then combined by equation (3.36) to obtain the complete second order solution to equation (3.27).

Note that the five-phase updating scheme (3.36) is in the order of $xyzyx$. In order to prevent systematic error resulting from a constant order of update, random ordering of update should be utilized. To be specific, we assign uniform probability to each of the following six order of update: $xyzyx$, $xzyzx$, $yxzxy$, $yzxzy$, $zxyxz$, and $zyxyz$.

3.2.3 Local Polynomial Fitting

The local polynomial fitting on arbitrary sets of points has long been used to obtain approximation of functions and their derivatives. Details of the method and its accurracy is discussed in [25, 27, 28]. Generally, νth order derivative can be approximated with $(n - \nu + 1)$th order of accuracy using nth order polynomial. For simplicity, a 2D example is discussed here. In the vicinity of a point 0, the function value in the location of a point i can be expressed by the Taylor series as

$$U_i = U_0 + h_i \left.\frac{\partial U}{\partial x}\right|_0 + k_i \left.\frac{\partial U}{\partial y}\right|_0 + \frac{1}{2}\left(h_i^2 \left.\frac{\partial^2 U}{\partial x^2}\right|_0 + k_i^2 \left.\frac{\partial^2 U}{\partial y^2}\right|_0 + 2h_i k_i \left.\frac{\partial^2 U}{\partial x \partial y}\right|_0 \right) + \ldots \tag{3.40}$$

where, U_i and U_0 are the corresponding function values in the location of points i and 0, $h_i = x_i - x_0$, $k_i = y_i - y_0$, and the derivatives are calculated in the location of the point 0. A polynomial can be used to approximate the original function and we employ a second order polynomial in this example:

$$\tilde{U} = U_0 + h_i\theta_1 + k_i\theta_2 + \frac{1}{2}h_i^2\theta_3 + \frac{1}{2}k_i^2\theta_4 + h_i k_i \theta_5 \tag{3.41}$$

Here, the variables θ_1, θ_2, θ_3, θ_4 and θ_5 are the estimates for $\frac{\partial U}{\partial x}$, $\frac{\partial U}{\partial y}$, $\frac{\partial^2 U}{\partial x^2}$, $\frac{\partial^2 U}{\partial y^2}$, and $\frac{\partial^2 U}{\partial x \partial y}$, respectively. In order to compute values of these variables, we perform a local polynomial fitting using $m >= 5$ points in the vicinity of

center point 0. The following linear system $Ax = b$

$$\begin{bmatrix} h_1 & k_1 & \frac{1}{2}h_1^2 & \frac{1}{2}k_1^2 & h_1k_1 \\ h_2 & k_2 & \frac{1}{2}h_2^2 & \frac{1}{2}k_2^2 & h_2k_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h_n & k_n & \frac{1}{2}h_n^2 & \frac{1}{2}k_n^2 & h_nk_n \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} U_1 - U_0 \\ U_2 - U_0 \\ \vdots \\ U_n - U_0 \end{bmatrix}, \tag{3.42}$$

is usually overdetermined. As a proper selection of a neighborhood is important for accuracy and stability, neighbor search algorithms used in our upwind solvers are described in the next subsection.

An optimal solution to (3.42) is a solution x that minimizes the L_2 norm of the residual, i.e.,

$$\min_x \|Ax - b\|_2, \tag{3.43}$$

and the QR decomposition with column pivoting is employed to obtain x. Suppose

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} P^T, m \geq n, \tag{3.44}$$

where Q is an orthonomal matrix, R is an upper triangle matrix, and P is a permutation matrix, chosen (in general) so that

$$|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{nn}|. \tag{3.45}$$

Moreover, for each k,

$$|r_{kk}| \geq \|R_{k:j,j}\|_2 \tag{3.46}$$

for $j = k+1, \cdots, n$. One can numerically determine an index k, such that the leading submatrix R_{11} in the first k rows and columns is well conditioned and R_{22} is negligible:

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix} \simeq \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} \tag{3.47}$$

Then k is the effective rank of A. Discussion about the numerical rank determination can be found in [29]. A simple way to determine numerical rank is to set a tolerance ϵ and find the first k such that

$$R_{kk} < \epsilon R_{11}. \tag{3.48}$$

If there is such k, then the effective numerical rank is $k-1$. The choice of ϵ is set to be 10^{-3} or 10^{-2} in the numerical examples presented in later sections. Note that simulation results are sensitive to the choice of ϵ in some cases (see section 3.2.4 for examples). In general, based on the dynamic algorithm for stencil selection, which is described in section 3.2.4, it often requires more neighbours to obtain effective rank in the case of larger ϵ, but the spatial derivatives calculated with larger ϵ's are often more robust.

The solution for linear system (3.42) can be obtained as

$$x = P \begin{bmatrix} R_{11}^{-1} c_1 \\ 0 \end{bmatrix} \tag{3.49}$$

where c_1 is the first k elements of $c = Q^T b$. This can also be written as

$$x = A^+ b \tag{3.50}$$

where

$$A^+ = P \begin{bmatrix} R_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T \tag{3.51}$$

is the pseudoinverse of matrix A.

3.2.4 The neighbor Search Algorithm and Dynamic Stencil Selection

In a simulation involving N Lagrangian particles, a new *stencil* of neighbors - those particles used for solving equation (3.42) - is to be selected at the begining of each time step for all the N particles. As a result, it is critical that an efficient neighbor search algorithm is employed. The neighbor search method is briefly introduced in the first subsection. While the neighbor search is to obtain a group of particles lying within some pre-specified distance away from the particle of interest, it is not necessary that all these neighbors are used in numerical stencils. The selection process of stencil points from neighbors to ensure the accuracy and stability is discussed in the second subsection.

Neighbor Search Algorithms

One of the main advantagees of the Lagrangian particle method compared to grid-based methods is its ability to simulate large and extremely non-uniform

domains. By a non-uniform domain we mean a domain in which only a small fraction of the total volume occupied by matter, found typically in astrophysics and high energy density physics, and other applications dealing with dispersed fragments of matter. For these applications, we use 2^k-tree neighbor search algorithms [30]. The 2^k-tree is a tree data structure in a k-dimensional space in which each node has at most 2^k dependents. Quadtree and octree are the standard terms in 2D and 3D spaces, respectively. The tree construction can be performed with $O(N \log N)$ operation, N being the total number of particles. In this process, the choice of the tree depth is essential and the optimal empirical number is four or five. After the construction step, the search of a tree for obtaining the neighborhood of a particle can be performed with $O(\log N)$ operation. The neighbour search algorithm based on quadtree/octree in the current code is developed by [45, 46] and is implemented by my teammates Gaurish Telang and Kwangmin Yu.

However the 2^k-tree method is not universally optimal for all types of problems. If the computational domain is almost uniformly filled with a weakly compressible matter in which inter-particle distances change insignificantly during the simulation allowing the use of the same neighbor search radius for all particles, the search of neighbors of a particle can be performed in constant time using algorithms such as the bucket search algorithm [17]. Specifically, in the bucket search algorithm the entire computational domain is divided into square (cubic) cells of the side length equal to two times the search radius r. For each particle inside a cell, only the neighboring cells need to be considered in the search process. For instance, 9 and 27 cells, which include the cell in

which the particle of interest resides, should be searched for neighbours in the 2D and 3D cases, respectively. Assuming that the number of particles within each cell can be approximated by a constant upper bound M, the neighbour search cost of a particle is therefore $9M$ and $27M$ in 2D and 3D, respectively. Clearly, the bucket search method is not applicable if different search radii must be used for different particles. The 2^k-tree neighbor search algorithm is more universal and applicable to a wide range of problems.

Dynamic stencil selection

After the neighbor search step, each particle obtains a list of neighbors which lie within the range of a pre-specified search radius. To enforce upwinding, however, only one-sided information should be used when solving equation (3.42). For the calculation of one-sided derivatives, each particle will, in general, have four one-sided neighborhoods in two-dimensions, and six one-sided neighborhoods in three-dimensions.

Without loss of generality, the process of the dynamic stencil selection will be discussed using an example of computing u_{xr}. After gathering one-sided neighbors, two main issues must be resolved for accurate evaluation of spatial derivatives. The first one is related to the *shape* of the stencil. To begin with, the list of one-sided neighbors is sorted by their distance from the center particle in ascending order (i.e. the neighbour which is nearnest to the center particle is sorted to be the first in the list). Suppose we obtain the following sorted list of 9 one-sided neighbors of the particle 0 (i.e. the 9 particles which reside within the disk with pre-specified search radius and with center particle

0, and are to-the-right of particle 0) for computing u_{xr}:

$$\{p_{1u},\, p_{2u},\, p_{3l},\, p_{4u},\, p_{5u},\, p_{6u},\, p_{7l},\, p_{8u},\, p_{9l}\}\,.$$

Here the subscripts u and l indicate the upper and lower half-planes in the y-direction: $y_i >= y_0$ and $y_i < y_0$, respectively. If a simple distance-based algorithm picks up six neighbors, then the corresponding stencil is composed of

$$\{p_{1u},\, p_{2u},\, p_{3l},\, p_{4u},\, p_{5u},\, p_{6u}\}\,,$$

thus producing a highly *unbalanced* stencil in terms of the *shape*. Therefore, besides sorting neighbors in ascending order of the distance from the center particle, the order is rearranged such that neighbors from the upper half and lower half occur interchangeably in the list

$$\{p_{1u},\, p_{3l},\, p_{2u},\, p_{7l},\, p_{4u},\, p_{9l},\, p_{5u},\, p_{6u},\, p_{8u}\}$$

The six-neighbor stencil now becomes:

$$\{p_{1u},\, p_{3l},\, p_{2u},\, p_{7l},\, p_{4u},\, p_{9l}\}$$

This approach yields more balanced-in-shape stencils, and typically results in more accurate spatial derivatives.

The second issue is the optimization of the *number* of neighbors for solving equation (3.42). In the case of second order local polynomial fitting in 2D,

for example, five neighbors are required to solve equation (3.42). However, as equation (3.48) suggests, the effective rank of the matrix A in equation (3.42) may be less than five if

$$R_{kk} < \epsilon R_{11},\ k = 2, 3, 4, 5 \tag{3.52}$$

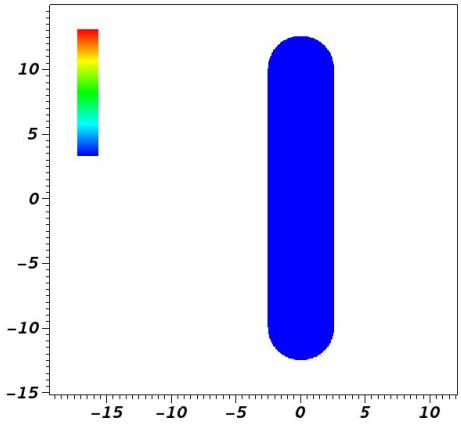
To avoid rank deficiency, a dynamic process for selecting neighbors into the stencil is designed. First, select the *tolerance* parameter ϵ as in equation (3.48). For the case of second order local polynomial fitting in 2D, one starts with six or seven neighbors in the stencil. Based on this stencil, the QR decomposition with column pivoting is calculated. Then determine the effective rank by equation (3.48). If the effective rank is no less than five the stencil is completed. Otherwise, the next neighbor in the neighbor list is added to the stencil. The process continues until the effective rank is regained. For instance, in some cases one may need to use eight neighbors to obtain an effective rank of five, depending on the *shape* of the stencil and also the tolerance parameter ϵ.

In cases that we have run out of neighbours for a particle in a specific direction, but still have not obtained the effective rank of the desired order of polynomial fitting, the order of local polynomial fitting is lowered for this particle in the given direction. For instance, suppose in 2D that a partcile has only six neighbours to the right in the x-direction, and using all of the six neighbours does not attain the required effective rank of 5 (for second-order polynomial fitting). Lowering to first order local polynomial fitting requires only an effective rank of two. As a result, we start by selecting, say 3 neighbors,

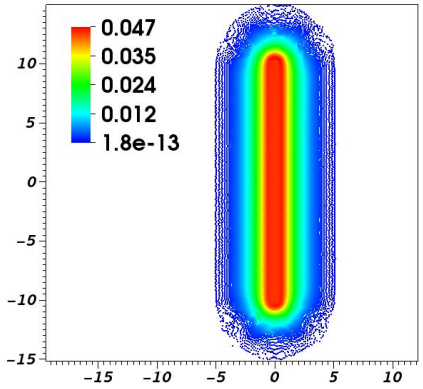
into the stencil and repeat the dynmic stencil selection process - for first-order polynomial fitting of this particle to the right in the x-direction. Once effective rank is ensured we are done. The worst case occurs when effective rank is not attained even for first order polynomial fitting. In such a case, we set all spatial derivatives to zero for this particle to the right in the x-direction.

One important thing to note is that the dynamic stencil selection process *remembers* the order of local polynomial fitting in a time step. To be specific, in a time step suppose the order of update of the Strang splitting is $xyzyx$ (3D case). Suppose during the first update in the x-direction (the first phase), the order of local polynomial fitting is lowered to the first order on the right, and remain second order on the left, due to the process of dynmaic stencil selection described above. For the second update in the x-direction (the fifth phase), we will start the dynamic stencil selection from first order on the right. The reason for memorizing the order of local polynomial fitting is obvious - the neighborhood is the same during each time step, so does the effective rank of the data matrix (the matrix A in equation (3.42)) in each direction.

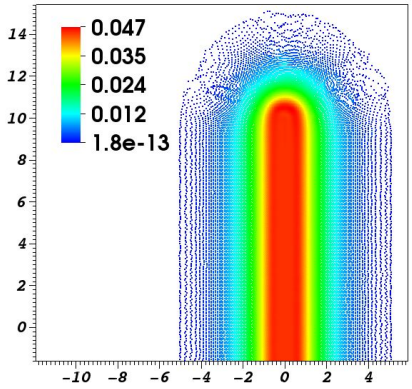
Note that simulation results may be sensitive to the value of the tolerance parameter ϵ. Figure 3.1 shows the results of the gas jet expansion into vacuum (the details are described in section 3.4.4 and see Figure 3.1(a) for the geometric configuration of the gas jet at time 0). Clearly, the choice of $\epsilon = 10^{-2}$ generates better results (spatial derivatives) than the choice of $\epsilon = 10^{-3}$ since the gas/vacuum interface is much smoother in the former case. This may due to the reason that in the case of $\epsilon = 10^{-2}$, more neighbours are often required to obtain the effective rank. It does not mean that using more neighbours



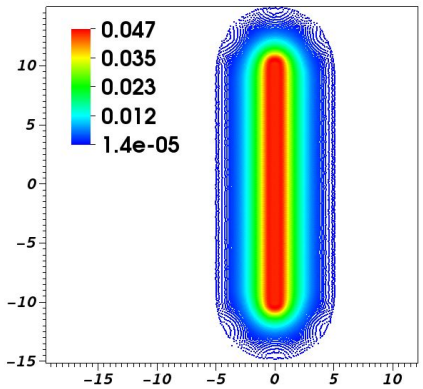
(a) t = 0.000 ms, $p = 4.873\ 10^{-2}$ (bar)



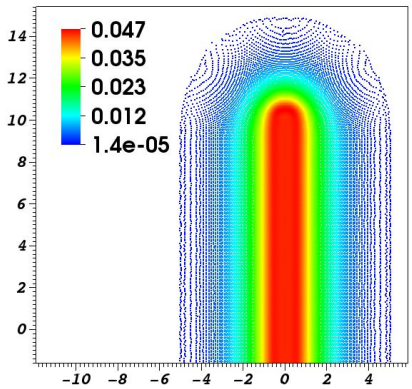
(b) t = 0.007 ms, $\epsilon = 10^{-3}$



(c) t = 0.007 ms , $\epsilon = 10^{-3}$, (zoomed-in)



(d) t = 0.007 ms, $\epsilon = 10^{-2}$



(e) t = 0.007 ms , $\epsilon = 10^{-2}$, (zoomed-in)

Figure 3.1: Pressure value of the gas jet simulation for two different ϵ's.

always generates more robust derivatives, since using too many neighbours will certainly smooth out the results. It simply says that using few neighbours may produce less stable results. Note, however, that our algorithm uses much smaller number of neighbor particles compared to the modified versions of SPH such as Godunov-SPH, P-SPH, PHANTOM etc. that may require hundreds of particles [21].

Variable Neighbour Search Radius

In the simulations of weakly compressible fluid, the inter-particle distance for all particiles is relatively a constant. As a result, we can use a constant value as the neighbour search radius for all particles during the entire simulation process. As a result, for weakly compressible fluid, the number of neighbours for most particles is relatively constant. This indicates that a constant value for the neighbour search radius makes most particles continue to have enough neighbours through the entire simulation process.

On the other hand, in simulations of highly compressible gas or fluid, the inter-particle distance between particles changes in several orders of magnitudes during the simulation process. For instance, in the gas jet expansion presented in section 3.4.4, the number density of particles changes from uniform to extremely non-uniform with particles near the center of the jet having much larger number density than those on the free surface of the expansion. Therefore, it is necessary to update the radius of neighbour search for all particles based on their own local number densities, because using a constant value of search radius gives too many neighbours for particles in dense areas,

while too less neighoburs for those in sparse regions. Note that based on the dynamic stencil selection process described in section 3.2.4, having less than necessary neighbours may lead to lowering the order of local polynomial fitting when computing spatial derivatives.

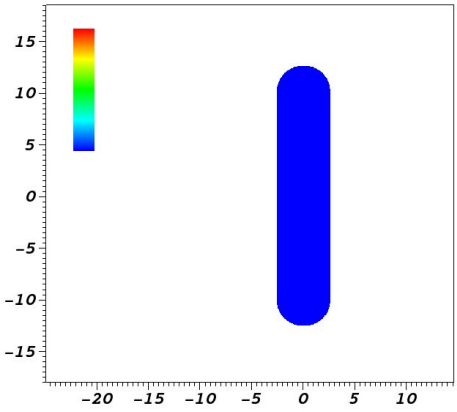
We propose using the specific volume to update the neighbour search radius for each particle at the end of every time step. To be specific, we propose updating the neighbour search radius of particle of index i (r_i) based on the following formula:

$$r_i^{t+1} = r_i^t \left(\frac{V_i^{t+1}}{V_i^t} \right)^{\frac{1}{3}} \tag{3.53}$$

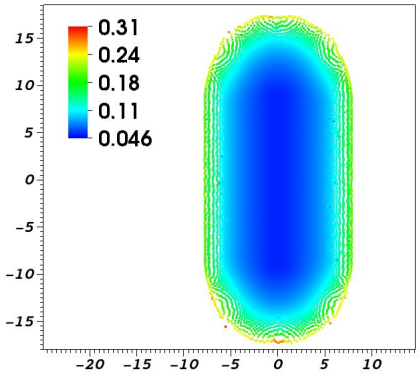
where the superscript t denotes the t-th time step and V denotes the specific volume. Figure 3.2 plots the updated values of inter-partcile spacing of the simulation of the gas jet expansion into vacuum at time 0.0125 (ms) (see section 3.4.4 for details). Note that the value of inter-particle spacing (sp_i) for each particle of index i is some constant fraction of the value of the neighbour search radius (r_i). For example, in the simulations presented in this thesis, $r_i = C\, sp_i$ where $C = 3$ or 4.

3.2.5 Limiters

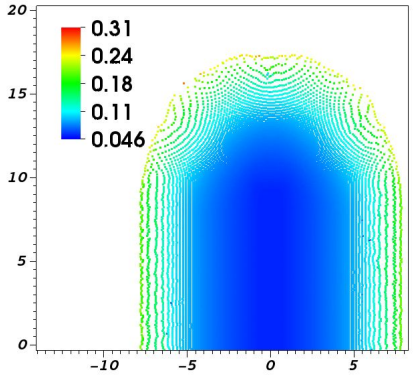
The second order Lagrangian particle algorithm is dispersive. To eliminate the resulting oscillations, two new types of limiters: limiter based on divided difference and flux-limiter were developed and coupled with the numerical integration. The applications of the algorithm with the limiters are demonstrated in section 3.4.



(a) t = 0.000 ms, inter-particle spacing = 0.05



(b) t = 0.0125 ms



(c) t = 0.0125 ms , (zoomed-in)

Figure 3.2: Inter-particle spacing of the simulation of jet expansion into vacuum.

The Limiter Based on Divided Difference

Since the source of dispersion comes from regions containing high gradients or discontinuities, it is desirable not to include information from these regions when calculating the spatial derivatives. This is accomplished by removing neighbors from regions containing discontinuities and using a reduced order interpolating polynomial with smaller number of stencil points, thus introducing diffusion instead of dispersion locally in the regions containing discontinuities.

Similarly to [31], we use the divided differences to detect the region that contains discontinuities. However, while the method of divided differences is used in [31] for choosing one of different stencils, it is employed in our work for switching between high order and low order spatial derivatives. The k-th order divided difference of function value V between the two particles at locations $x_{j-\frac{1}{2}}$ and $x_{j-\frac{1}{2}+k}$, $k \geq 1$, is defined recursively as follows.

$$V\left[x_{j-\frac{1}{2}}, \ldots, x_{j-\frac{1}{2}+k}\right] \equiv \frac{V\left[x_{j+\frac{1}{2}}, \ldots, x_{j-\frac{1}{2}+k}\right] - V\left[x_{j-\frac{1}{2}}, \ldots, x_{j-\frac{3}{2}+k}\right]}{x_{j-\frac{1}{2}+k} - x_{j-\frac{1}{2}}}. \tag{3.54}$$

For the case of $k = 1$,

$$V\left[x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}\right] \equiv \frac{V\left(x_{j+\frac{1}{2}}\right) - V\left(x_{j-\frac{1}{2}}\right)}{x_{j+\frac{1}{2}} - x_{j-\frac{1}{2}}}. \tag{3.55}$$

And for the case $k = 2$,

$$V\left[x_{j-\frac{1}{2}}, \ldots, x_{j+\frac{3}{2}}\right] \equiv \frac{V\left[x_{j+\frac{1}{2}}, x_{j+\frac{3}{2}}\right] - V\left[x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}\right]}{x_{j+\frac{3}{2}} - x_{j-\frac{1}{2}}}. \tag{3.56}$$

One important property of divided differences is expressed as

$$V\left[x_{i-\frac{1}{2}},\ldots,x_{i-\frac{1}{2}+k}\right]=\frac{V^{(k)}(\xi)}{k!} \tag{3.57}$$

for some ξ inside stencil $x_{i-\frac{1}{2}}<\xi<x_{i-\frac{1}{2}+k}$, for some $k\geq 1$, as long as the function $V(x)$ is smooth in this stencil. If $V(x)$ is discontinuous at some point inside the stencil, then

$$V\left[x_{i-\frac{1}{2}},\ldots,x_{i-\frac{1}{2}+k}\right]=O\left(\frac{1}{\Delta x^k}\right). \tag{3.58}$$

As a result, a smaller divided difference indicates that the function $V(x)$ is smoother in a stencil, thus providing a convenient measure for smoothness.

Let the pressure $p(x_j)$ at particle location x_j represent the average pressure in the interval $[x_{j-\frac{1}{2}},x_{j+\frac{1}{2}}]$. Define the cumulative pressure $P(x_j)$ as

$$\begin{aligned} P(x_{j+\frac{1}{2}})=P(x_{j-\frac{1}{2}})+p(x_j)\left(x_{j+\frac{1}{2}}-x_{j-\frac{1}{2}}\right),\ \text{for } j\geq 0,\\ P(x_{-\frac{1}{2}})\equiv 0. \end{aligned} \tag{3.59}$$

For $k\geq 1$, the divided difference of k-th order is computed recursively by

$$P\left[x_{j-\frac{1}{2}},\cdots,x_{j-\frac{1}{2}+k}\right]\equiv\frac{P\left[x_{j+\frac{1}{2}},\cdots,x_{j-\frac{1}{2}+k}\right]-P\left[x_{j-\frac{1}{2}},\cdots,x_{j-\frac{3}{2}+k}\right]}{x_{j-\frac{1}{2}+k}-x_{j-\frac{1}{2}}}. \tag{3.60}$$

For $k\geq 1$, define the k-th degree divided difference for particle j *on the left*

hand side as

$$DD_j^{\text{left}}(k) \equiv \frac{P\left[x_{j+\frac{3}{2}-k}, \ldots, x_{j+\frac{1}{2}}\right] - P\left[x_{j+\frac{1}{2}-k}, \ldots, x_{j-\frac{1}{2}}\right]}{x_{j+\frac{1}{2}} - x_{j+\frac{1}{2}-k}}. \tag{3.61}$$

Similarly, for $k \geq 1$, define the k-th degree divided difference for particle j *on the right hand side* as

$$DD_j^{\text{right}}(k) \equiv \frac{P\left[x_{j+\frac{1}{2}}, \ldots, x_{j-\frac{1}{2}+k}\right] - P\left[x_{j-\frac{1}{2}}, \ldots, x_{j-\frac{3}{2}+k}\right]}{x_{j-\frac{1}{2}+k} - x_{j-\frac{1}{2}}}. \tag{3.62}$$

For computational efficiency, one can compute the divided differences iteratively:

$$\begin{aligned} DD_j^{\text{left}}(k) &= \frac{DD_j^{\text{left}}(k-1) - DD_{j-1}^{\text{left}}(k-1)}{x_{j+\frac{1}{2}} - x_{j+\frac{1}{2}-k}},\ k \geq 1 \\ DD_j^{\text{right}}(k) &= \frac{DD_{j+1}^{\text{right}}(k-1) - DD_j^{\text{right}}(k-1)}{x_{j-\frac{1}{2}+k} - x_{j-\frac{1}{2}}},\ k \geq 1 \end{aligned} \tag{3.63}$$

And $DD_{j+q}(0) \equiv P(x_{j+\frac{1}{2}+q})$, q being any integral number.

The proposed limiter works as a switch between higher and lower order schemes to avoid the oscillatory behavior. Let us examine in more detail the left-hand-side example. To choose between $k-1$th and $k-2$th order of accuracy, we compute $DD_j^{\text{left}}(k)$. For a selected global constant C, if

$$DD_j^{\text{left}}(k) > C \tag{3.64}$$

we switch from $(k-1)$th order to $(k-2)$th order derivatives by excluding

the neighbor of index $j-(k-1)$ (i.e. the $k-1$th neighbor on the left-hand-side of particle j). For example, if $DD_j^{\text{left}}(3) > C$, we use the first-order local polynomial fitting for $u_{xl}(j)$, $p_{xl}(j)$, instead of the second-order one. In this case, $u_{xxl}(j)$, and $p_{xxl}(j)$ are assigned 0. The same logic applies to the right-hand-side derivatives.

The Flux-limiter

In the flux-limiter method, the magnitude of the correction depends on the smoothness of data (represented by Φ), and can be written as

$$F(U;j) = F_L(U;j) + \Phi(U;j)[F_H(U;j) - F_L(U;j)] \tag{3.65}$$

In order to measure the smoothness of data, we can use the ratio of consecutive gradients:

$$\theta_j = \frac{U_j - U_{j-1}}{U_{j+1} - U_j} \tag{3.66}$$

Or we can use the average of the ratio of consecutive gradients from both directions:

$$\theta_j = \frac{1}{2}\left(\frac{U_j - U_{j-1}}{U_{j+1} - U_j} + \frac{U_{j+1} - U_j}{U_j - U_{j-1}}\right) \tag{3.67}$$

Equation (3.67) has the advantage that it is symmetric. If θ_j is near 1 the data is presumably smooth. If θ_j is far from 1 there may be discontinuity near data U_j. Let $\Phi(U;j) \equiv \phi_j$ to be a function of θ_j:

$$\phi_j = \phi(\theta_j) \tag{3.68}$$

Van Leer [47] proposed a smooth limiter function

$$\phi(\theta) = \frac{|\theta| + \theta}{1 + |\theta|} \tag{3.69}$$

We let $\phi(\theta) = 0$ for $\theta < 0$ or when θ is arbitrarily large. Nore that $\theta < 0$ in the case when $U_{j+1} - U_j$ and $U_j - U_{j-1}$ are in opposite signs in both equations (3.66) and (3.67)). θ is arbitrarily large when $U_{j+1} - U_j = 0$ in equation (3.66) and when $U_{j+1} - U_j = 0$ or $U_j - U_{j-1} = 0$ in equation (3.67).

Without loss of generality, we demonstrate the idea using the flux for volume. Remind that in the proposed Lagrangian particle method the volume flux is defined as (equation (3.22))

$$\begin{aligned} V_t = & \frac{V}{2}\left(u_{xr} + u_{xl}\right) - \frac{V}{2\sqrt{K}}\left(P_{xr} - P_{xl}\right) \\ & + \frac{\Delta t}{4}\left[V^2\sqrt{K}\left(u_{xxr} - u_{xxl}\right) - V^2\left(P_{xxr} + P_{xxl}\right)\right] \end{aligned} \tag{3.70}$$

Let the lower order flux (first order flux) of volume be defined as

$$F_L(V) = \frac{V}{2}\left(u_{xr(1)} + u_{xl(1)}\right) - \frac{V}{2\sqrt{K}}\left(P_{xr(1)} - P_{xl(1)}\right) \tag{3.71}$$

where the subscript (1) denotes the spatial derivatives obtained by first order polynomial fitting. Then define the higher order flux (second order flux) of

volume be defined as

$$F_H(V) = \frac{V}{2}\left(u_{xr(2)} + u_{xl(2)}\right) - \frac{V}{2\sqrt{K}}\left(P_{xr(2)} - P_{xl(2)}\right) + \frac{\Delta t}{4}\left[V^2\sqrt{K}\left(u_{xxr(2)} - u_{xxl(2)}\right) - V^2\left(P_{xxr(2)} + P_{xxl(2)}\right)\right] \tag{3.72}$$

where the subscript (2) denotes the spatial derivatives obtained by second order polynomial fitting.

In order to make the measure of the smoothness of data (θ) generalizable to higher dimensions and applicable to the Lagrangian particle mehtod, we propose using the one-sided spatial derivatives calculated by methods introduced in section 3.2.3. Depneding on the type of data we have, θ of a particle j is calculated as:

$$\theta_j(u) = \frac{u_{xl(1)}}{u_{xr(1)}} \tag{3.73}$$

$$\theta_j(P) = \frac{P_{xl(1)}}{P_{xr(1)}} \tag{3.74}$$

where u is the velocity in the x, y, or z-direction and P is the pressure. Alternatively, we can also use

$$\theta_j(u) = \frac{1}{2}\left(\frac{u_{xl(1)}}{u_{xr(1)}} + \frac{u_{xr(1)}}{u_{xl(1)}}\right) \tag{3.75}$$

$$\theta_j(P) = \frac{1}{2}\left(\frac{P_{xl(1)}}{P_{xr(1)}} + \frac{P_{xr(1)}}{P_{xl(1)}}\right) \tag{3.76}$$

Note that equations (3.75) and (3.76) are better over (3.73) and (3.74) since

they are symmetric. We choose

$$\theta_j = \min(\theta_j(u), \theta_j(P)) \tag{3.77}$$

and calculate ϕ_j by (3.69). Note that we set $\phi_j = 0$ when $\theta_j < 0$ or θ_j is arbitrarily large. Substituting the calculated ϕ_j, the lower and higher flux in (3.71) and (3.72) into (3.65), we obtain the volume flux of particle j as:

$$F_j(V) = F_L(V; j) + \phi_j[F_H(V; j) - F_L(V; j)] \tag{3.78}$$

Then time integration gives the volume at next time step as:

$$V^{n+1} = V^n + \Delta t F_j(V) \tag{3.79}$$

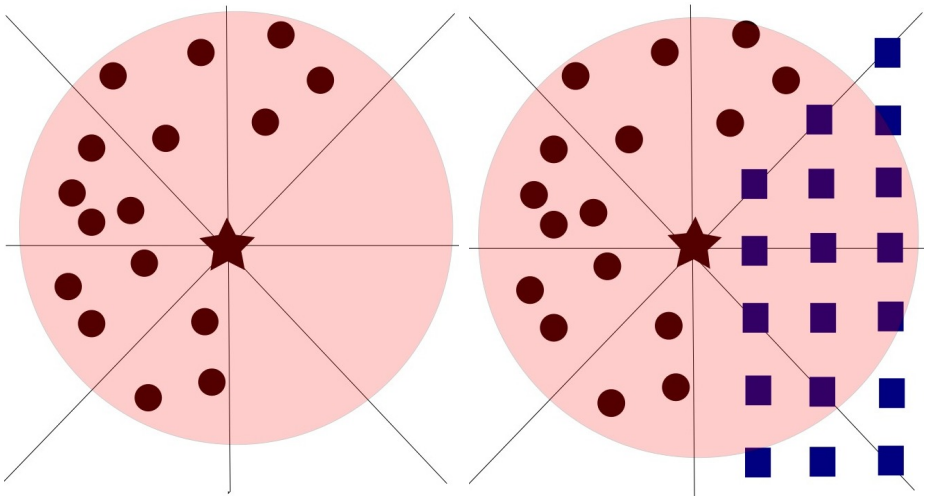
3.2.6 Modelling of Free Surfaces using Ghost Particles

An important feature of the Lagrangian particle method is its ability to robustly handle free surface flows with geometrically complex interfaces. The method is also generalizable to multiphase problems. Here by free surface flows we mean flows of fluid or gas in vacuum, and by multiphase problem we mean the interface dynamics between two immiscible fluids or gases. In this section, we describe an algorithm for physically consistent solutions at free fluid or gas interfaces.

In our method the fluid/vacuum interface is modeled by using *ghost* particles in the vacuum region. An geometric algorithm has been developed to

create patches of ghost particles for each fluid particle in the vacuum region. The local vacuum region of a fluid particle, is determined by its local neighbourhood. For example, in 2D, we divide the local neighbourhood into 8 subdomains as shown in Figure 3.3. In Figure 3.3(a) the original local neighbourhood (marked as a fainted circle region centered at the star) of a fluid particle (marked as a star) is plotted. Since there are 3 empty regions with no fluid neighbours (marked as dots), these 3 empty regions are identified as vacuum regions and ghost particles (marked as squares) are filled inside these 3 regions. In the end, only those ghost particles lying inside the local neighbourhood are generated for this fluid particle. In 3D, the same algorithm is applied with local neighbourhood divided into 16 regions. Note that these ghost particles are used only by the one corresponding fluid particle (by which they are generated), for the calculation of spatial derivatives.

After identifying the location of ghost particles their physics states are assigned. Since the only use of ghost particles is to serve as neighbors of fluid particles when calculating spatial derivatives, only two states are relevant: pressure and velocities. As ghost particles represent vacuum, their pressure state is assigned to zero. For the assignment of velocities, we propose two methods. First, since ghost particles move with the fluid/gas, the ghost particles can simply be assigned the same velocities as the corresponding fluid particles. The second method is based on a weighted 0th order local polynomial fitting. To be specific, we compute weighted average of velocities of the fluid particles that are in a neighborhood of the ghost particle. Let's assume that the weighting function of the particle 0 in a three-dimensional space is



(a) Original neighbourhood (b) After ghost particles are filled

Figure 3.3: The local neighbourhood (fainted circle region) of a fluid particle (star). The neighbour hood is divided into 8 regions. Inside each region, if no fluid neighbours (dots) are found, ghost particles (squares) are filled. Only those ghost particles lying inside the local neighbourhood are generated.

$w(h_j, k_j, g_j)$, where $h_j = x_j - x_0$, $k_j = y_j - y_0$, $g_j = z_j - z_0$, and j is the index of neighbor particles. The velocity u_0 of particle 0 satisfies

$$\min_{u_0} \sum_{j=1}^{N} \left[(u_0 - u_j) w(h_j, k_j, g_j)\right]^2, \tag{3.80}$$

which leads to the solution

$$u_0 = \frac{\sum_{j=1}^{N} u_j w_j^2}{\sum_{j=1}^{N} w_j^2} \tag{3.81}$$

In Figure 3.4 we plot the simulation results of the gas jet expansion into vacuum at time 0.0025 (ms) using these two methods. Note that the two methods produce similar results, while the second one expands at a slightly

slower velocity with a more smoothed direcitons of expansion, as expected. Note that the two methods were adopted after comparing them with a more sophisticated one based on a solution of the Riemann problem for boundary particles in the direction normal to the interface. Our conclusion was that the costly Riemann solver algorithm is not required for the fluid/vacuum interface (but will be the most likely choice for the interface in multiphase problems).

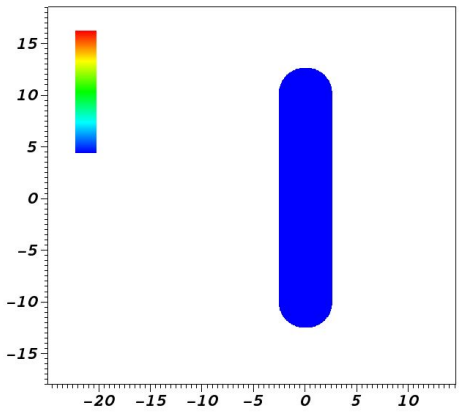
3.3 Lagrangian Particle Hydrodynamics Code

In previous sections, the theory of the proposed Lagrangian particle method for solving Euler equations and the related numerical algorithms have been described. We have developed a corresponding code which implements the algorithms and are capable of running 1D/2D/3D simulations. The construction of the code strictly follows the idea of object-oriented design (OOD) principles:

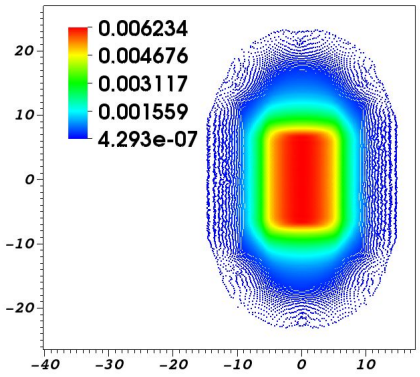
1. The code divides the Lagrangian particle method into major tasks and model the tasks using classes. Therefore, the code is conceptually clear and easy to maintain.

2. The code models algorithms that may permit alternative solutions using ploymorphic classes and well-designed public interface. In this manner, future extensions of the algorithm can be done simply by constructing derived classes of the parent class and implementing the alternative solution by overriding the virtual methods.

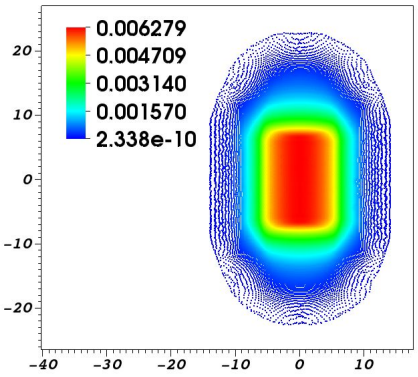
3. The code hides data and implementation details in *private* sections of



(a) t = 0.000 ms, $p = 4.873\ 10^{-2}$ (bar)



(b) t = 0.0025 ms



(c) t = 0.0025 ms

Figure 3.4: Pressure value of the simulation of jet expansion into vacuum. In Figure 3.4(b) ghost particles are assigned the same velocities as their corresponding fluid particle. In Figure 3.4(c) ghost particles are assigned the weighted average velocities of its neighbouring fluid particles as computed by equation (3.81).

the class (implementation hiding). As a result, it is possible to correct or improve old implementations, or adding data to classes, without affecting the client code.

The implementation of the code put ultimate emphasis on time-efficiency (over space-efficiency) by a judicious selection the data structure (see 3.3.3) and very efficient implementations (for example, see 3.3.7). For 3D capabilities, it is necessary to parallelize the code and the current has been parallelized using shared-memory parallelization techniques based on the OpenMP API. The current code is written in C++ and has about $20,000$ lines of code. I put the code on my GitHub, and my GitHub page and the code documentation page is located at

- https://github.com/HsinchiangChen/LPFluidCode
- http://hsinchiangchen.github.io/LPFluidCode/index.html

Figure 3.5 depicts the object-oriented design structure and the work flow of the code. In the subsequent subsections, we describe how the work in the proposed algorithm is divided into independent units based on C++ classes. We will also describe the data structures and implementations chosen for each task.

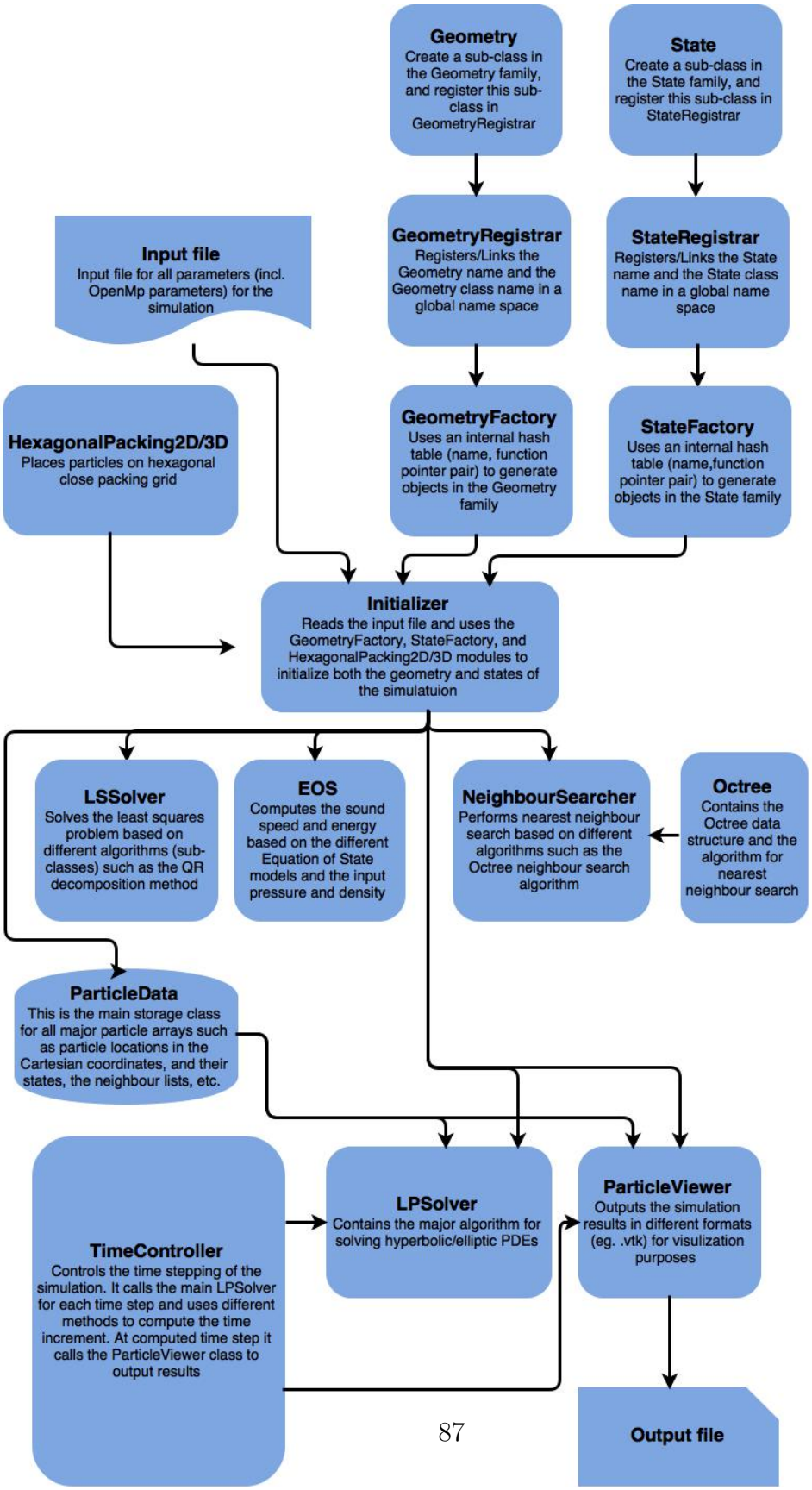


Figure 3.5: The object-oriented design structure and the work flow of the code.

3.3.1 The Geometry and State Classes and User Inputs for Initialization

In this section we will show the procedures for the user of the code to set up the initial settings of a specific simulation, and how the code handles these settings. In general, the initialization of any simulation using the Lagrangian particle algorithm involves three tasks, namely, initializing the particle geometry (particle location in the Cartessian coordinates), initializing particle states such as pressure, density, and velocities in the x, y, and z-directions, and initialzing other parameters such as the choice of EOS, the physical end time, etc.

The family of *Geometry* classes takes care of the first task by overloading the operator() and using it as a level set function. The function prototype is (in the base class *Geometry*)

```
virtual bool operator()(double x, double y, double z)=0;
```

returns true if (x, y, z) is within the spatial doamin of the defined geoemtry. That is, the user needs to create a child class of the *Geometry* class and override this function by specifying the level set function of the desired geometry. For example, the level set function of a ball with center 0 and radius r in 3D can be sepcified as

$$\text{return } x^2 + y^2 + z^2 <= r^2.$$

The seond task is initializing particle states, which is handled in the family of *State* classes: The following functions in the *State* class return the pressure,

density, and velocities in the x, y, and z-directions:

```
virtual double pressure(double x, double y, double z)=0;
virtual double density(double x, double y, double z)=0;
virtual void velocity(double x, double y, double z, double& vX,
    double& vY, double& vZ)=0;
```

Then the user of the code can initialize the particle geometry and states of the desired simulation by writting subclasses of the classes *Geometry* and *State* and overriding these pure virtual functions.

The third task is creating an input file to specify parameters, such as the choice of the geometry and state for the simulation. The class *Initializer* reads the input file and is responsible for allocating memory and initialize data arrays. (see section 3.3.2 for details). It becomes necessary to modify code in the *Initializer* class when, say, different geometries are used. For example, changing the geometry from a Circle to a Triangle in 2D requires changing the code in *Initializer* from

```
Geometry* gp = new Circle();
```

to

```
Geometry* gp = new Triangle();
```

This sort of code modification is error-prone and should be avoided. We prevent the modification by employing the Abstract Factory design pattern: creating factories for the *Geometry* and *State* classes, which are implemented in the classes of *GeometryFactory*, *StateFactory*, and helper classes *GeometryReg-*

istrar, and *StateRegistrar*. Using factories the code becomes

```
Geometry* gp = GeometryFactory::instance().createGeometry(gname);
State* sp = StateFactory::instance().createState(sname);
```

where "gname" and "sname" are strings specified in the input file, representing the names of the desired particle goemetry and states. Note that these strings (names geometry and state) are linked to the subclasses of *Geometry* and *State* through the helper template classes *GeometryRegistrar*, and *StateRegistrar*. For instance, the following codes link the name "ball" with the *Ball* class and the name "gauss" with the *GaussianPressureState* class.

```
GeometryRegistrar<Ball> r1("ball");
StateRegistrar<GaussianPressureState> s1("gauss");
```

It follows that changing geometries or states requires only changing names specified in the input file (and creating subclasses of the *Geometry* and *State* if they do not exist yet, and link the names with classes as we did above).

To summarize, initialization of particle geometry and states of the desired simulation in general requires three steps:

1. Derive from the *Geometry* and *State* classes, and override the level set functions and the functions for states.

2. Register/Link the names of the created classes in Registrar.cpp.

3. Put the names of the desired geoemtries and states in the input file.

3.3.2 The Initializer Class

The *Initializer* class initializes the simulation by

1. Reading the input file.

2. Creating *Geometry* and *State* objects, and utilizing them to allocate memory of data arrays such as the particle location in the Cartessian coordinates ((x,y,z)) and pressure, volume, velocities in the x, y, and z-directions.

3. Forwarding data arrays or parameter values to other classes.

Note the *Initializer* class forwards the allocated memory (data arrays of location and states) to the class *ParticleData*, which stores major data of the simulation. It also serves as the input to the constructors of other classes to distribute information from the input file to other classes. Using the *Initializer* class as input not only prevents modification of the constructors of other classes while the number of the content of input parametes are changed, but it also simplifies the function prototypes.

3.3.3 The ParticleData class

The ParticleData class holds the majority of data used in the simulation. The original idea is to save all data, such as location, pressure and volume, into a single data structure as the following.

```
class ParticleData {
public:
```

```
  ... // getter of the private data members
private:
  double x, y, z;
  double pressure, volume, velocity_x, velocity_y, velocity_z;
  std::vector<int> neighbour_index;
  ... // other particle data
}
```

Then suppose there are a total of N particles, we will allocate memory using

```
ParticleData* parray = new ParticleData[N];
```

This design is intuitive in that it looks like each physical particle owns a single ParticleData instance. Nevertheless, this data structure is not the most efficient choice for the current code due to the data proximity problem. To be specific, the memory locations for the same quantity, say pressure, of the N particles, are not continguous. However, in order to read or update data, in the current code we usually use loops to "sequentially" go through a specific quantity of all N particles. For example, to update pressure and volume for all N particles, we write

```
for(size_t index=0; index<N; index++) {
  pressure[index] = aFunc(...);
  volume[index] = anotherFunc(...);
  ...
}
```

As a result, it will be much more efficient if we use simple one-dimensional arrays (or vectors in C++) to hold these particle data, because the memory is contiguous for such data structures. Therefore, the data stored in the *ParticleData* class are mostly 1D arrays:

```
class ParticleData {
public:
  ... // getter of the private data members
private:
  double* positionX;
  double* positionY;
  double* positionZ;
  double* velocityU;
  double* velocityV;
  double* velocityW;
  double* volume;
  double* pressure;
  double* soundSpeed;
  int* neighbourList;
  ... // other particle data

}
```

The length of these data arrays is uniformly a bit larger than the total number of fluid (and boundary) particles (N), say, $1.5N$. The additional space is left for ghost particles, whose number changes during the simulation. The

only exceptions are those data arrays storing neighbour indices, such as the neighbourList array in the above code. We set the length of neighbourList to be $(1.5N)K$, where K is a pre-specified number representing the upperbound of the number of neighbours a particle can possibly have, during the entire simulation. Note that for efficiency, these array memory is only allocated once at initialization. Updating information stored in these arrays rewrites these arrays, but never re-allocate them. It is true that there are usually spaces wasted, for example, the number of ghost particles at time 0 is M with $M << 0.5N$, or K is larger than the number of neighbours for most particles. However, the extra space left for the possible increase in the number of ghost particles, or the number of neighbours of a particle, is worthwhile since memory re-allocation is avoided - yielding a much better performance in terms of time efficiency.

3.3.4 The LSSolver Classes

The algorithm for solving the least squares problem has been discussed in section 3.2.3. The implementation is in the family of LSSover classes, which has the interface:

```
class LSSolver {
public:
  virtual int solve(double* result, double* b) = 0;
   ... // other functions
private:
```

```
    ... // some data
};
```

Currently, only the QR decomposition algorithm has been implemented in the subclass *QRSolver*, which is in fact a wrapper class of the LAPACK LAPACKE_dgeqp3() routine that implements the QR decomposition with column pivoting. Note that the constructor of the subclass, say the *QRSolver* class, needs to take the data matrix (the A matrix in equation (3.42)) before calling the solve() method. When calling the solve() method, the data array "b" (the b vector in equation (3.42)) is used as input, while the data array "results" will contain the calculated derivatives after the function returns. If solve() returns 0 then the spatial derivatives are used for time integration. However, it it returns non-zero values, the dynamic stencil selection process described in section 3.2.4 will be employed to re-calculate the matrix A. The process repeats until effective rank is attained, or the polynomial fitting order is lowered to 0, in which case the derivatives will be set to zero. Note that other polymorphic subclasses can be created for other algorithms for solving least-squares problem, such as SVD (see [29] for discussions on alternative solutions for solving least-squares problems).

3.3.5 The EOS (Equation of State) Classes

The family of EOS classes is to calculate the internal energy e and the sound speed cs based on values of pressure and specific volume (1/density). Different EOS models have different functions for calculating the internal energy (e) and

sound speed (cs) from pressure (P) and specific volume (V): $e = f(P, V)$ and $cs = g(P, V)$. It follows that the interface for the abstract class EOS is:

```
class EOS {
public:
  virtual double getEnergy(double pressure, double density) = 0;
  virtual double getSoundSpeed(double pressure, double density) = 0;
  ... // other functions
private:
   ... // some data
};
```

Currently two EOS models have been implemented, namely, the polytropic gas EOS (the *PolytropicGasEOS* class) and the stiffened polytropic gas EOS (the *StiffPolytropicGasEOS* class). Note that the calculation of sound speed is necessary in the proposed Lagrangian method only for the calculation of the size of the time step based on CFL conditions, while the internal energy is essentially not used anywhere in the method.

3.3.6 The NeighbourSearcher Classes

The family of NeighbourSearcher classes performs the nearest neighbour search. The interface of the abstract class NeighbourSearcher class is as follows.

```
class NeighbourSearcher {
public:
```

```
  virtual int buildSearchStructure(const double* x, const double*
      y, const double* z, size_t numParticles) = 0;
  virtual int buildSearchStructure(const double* x, const double*
      y, const double* z, size_t begin, size_t numParticles) = 0;
#ifdef _OPENMP
  virtual int searchNeighbour(const double x0, const double y0,
      const double z0, const double radius, int* result, double*
      distance, size_t& result_length, int tid, int index = -1) = 0;
#else
  virtual int searchNeighbour(const double x0, const double y0,
      const double z0, const double radius, int* result, double*
      distance, size_t& result_length, int index = -1) = 0;
#endif
  ... // other functions
private:
  ... // some data
};
```

Currently, the octree neighbour search algorithm, described in section 3.2.4 has been implemented (the *OctreeSearcher* and the *Octree* classes) by my teammates Gaurish Telang and Kwangmin Yu.

The method buildSearchStructure() builds an octree based on data arrays of particle locations (x, y, z). There are two versions of the buildSearchStructure() function to allow for the flexibility of using different portions of the data arrays as the input for octree building.

After the octree is built, we look for neighbours of a particle with index 0 by calling the function searchNeighbour() with the its location $(x0, y0, z0)$ and a search radius as the inputs. Note that the input search radius can be different for different particles. Let the distance from the particle 0 to its neighbour j be d_j, the output of the method is an array of neighbour indices j (result), the array of d_j (distance), and the length of the two arrays (result_length). THe neighbour list index array (result), when returned, has been sorted in ascending order of d_j, which accomplishes the first step in the dynamic stencil selction process described in section 3.2.4. There are two versions of the method searchNeighbour(), and the first one is the multithreaded version. In the multithreaded version, tid represents the thread index, and is used as an input to prevent data races.

Note that the method buildSearchStructure() does not have a multithreaded version since it contains OpenMP conditional compilation directives in its implementation. As a result, as long as the compilation is done with OpenMP enabled, and the number of threads specified in the input file is larger than one, the tree building will automatically be parallelized.

3.3.7 The LPSolver Classes

The family of LPSolver classes performs the simulation for one time step - it implements the time integration by Strang splitting and uses all the classes that implement the algorithms described in section 3.2. The interface of the LPSolver class is as follows.

```
class LPSolver {
public:
  virtual int solve(double dt) = 0;
  ...
private:
   ... // some data
};
```

Currently, only hyperbolic equations are solved and two classes have been implemented: one is the *HyperbolicSolver* class for the 2D and 3D cases, and the *HyperbolicSolver1D* class for the 1D case. The reason for using different classes based on dimensionality is that in 1D, the Strang splitting is not used, and many tasks such as neighbour search are not necessary. As a result, to avoid numerous unnecessary conditional expressions based on the dimensionality, we create a independent class for 1D. Besides, in many situations, the 1D code is usually the best choice for testing new ideas/algorithms. Therefore, it is better to save the 2D and 3D code from being modified when experimenting with tests.

The procedures in HyperbolicSolver can be summarized as follows.

1. Generate ghost particles based on the current configuration of fluid particles.

2. Perform neighbour search based on *NeighbourSearcher* classes.

3. Select one-sided neighobur list based on the neighbour list obtained in the previous step and the dynmaic stencil selection algorithm (the part

which is based on the shape of the shape of the neighbourhood) described in section 3.2.4.

4. Set the values of pressure and velocities for the ghost particles based on the algorithms described in section 3.2.6.

5. Calculate the minimum spacing between particles, the maximum sound speed of all particles, and the maximum speed of fluid flows. These values are used to compute the lenth of time step for this iteration based on the CFL criterion (by the *TimeController* classes).

6. Implement the time integration by the Strang directional splitting described in section 3.2.2. In each phase of the directional splitting, the spatial derivatives are computed by local polynomial fitting and the second part of the dynamic stencil selection, which depends on the number of neighbours and the effective rank, is performed. The least squares problem is solved by using the *LSSolver* classes.

3.3.8 The ParticleViewer Classes

The family of ParticleViewer classes is created to output results for visualization purposes. The interface of the abstract class ParticleViewer is as follows.

```
class ParticleViewer {
public:
  virtual int writeResult(double time, std::size_t writeStep) = 0;
   ... // other functions
```

```
 private:
    ... // some data
};
```

Currently, results obtained from two and three dimensions simulations are visualized using the software VisIt, and results are saved in .vtk format (the *VTKParticleViewer* clas). One dimensional results are saved in simple textfile formats (the *TXTParticleViewer1D* class). Polymorphic behaviors happen in that the writeResult() method output data in the vtk format in one case, and in the txt format in the other. The inputs of writeResult() are the physical time and the number of steps of outputting, respectively.

3.3.9 The TimeController classes

The family of TimeController classes controls the behavior between any two time steps of the simulation, which involves three major tasks.

1. Calculate the time step used for next time step. For example, the time step can be computed based on the CFL condition.

2. Solve the system of equations of the desired problem at the current time step. For example, we can solve the Euler equation by using the *LPSolver* classes.

3. Output data when necessary by using the *ParticleViewer* classes.

The interface of the TimeController class is:

```
class TimeController {
public:
   virtual int solve() = 0;
   ... // other methods
protected:
   LPSolver* solver;
   std::vector<ParticleViewer*> viewers;
private:
   ... // some data
};
```

Each different combination of the choices of algorithms for the three tasks listed above may give one new type of *TimeController*. Currently, only one type, which follows the algorithm described in section 3.2, has been implemented: the time step is calculated based on the CFL condition, and the Euler equations (equations (3.1) and (3.2)) are solved. In the future we may solve coupled systems of hyperbolic and elliptic equatuions by creating a new subclass. In general, we can add or modify any features between every two time steps of the simulation of interest by creating new subclasses of the *TimeController* class.

3.3.10 Scalability Tests

The code has been parallelized using the shared-memory OpenMP model. We demonstrate the scalability of the current code by running the simulation of 3D

gas jet expansion into vacuum. (see section 3.4.4 for the detailed simulation settings). The initial inter-particle spacing (resolution) is 0.08 (cm), which results in a total of 723049 fluid particles. We run the simulation using the serialized version of the code, and the parallelized version using 4, 8, and 12 threads, and plot the speedups in Figure 3.6. Note that the speedups are measured in terms of the time spent on the corresponding function during one entire time step, with the dots denote the speedups for the entire time step, the squares denote the speedups in the directional splitting routine, and the stars denote the speedups in the neighbour search routine. This result demonstrate that the code scales almost linearly. Figure 3.7 plots the time (on which the speedups in Figure 3.6 are computed) spent on these routines. It is clear from this plot that these two routines constitute more than 90 peprcent of the total time (i.e. the time spent on all other functionalities during the simulation are irrelevant).

3.4 Numerical Results

In this section, we present results of one- and two-dimensional simulation that serve as verification tests for the Lagrangian particle method, including the free surface algorithm,

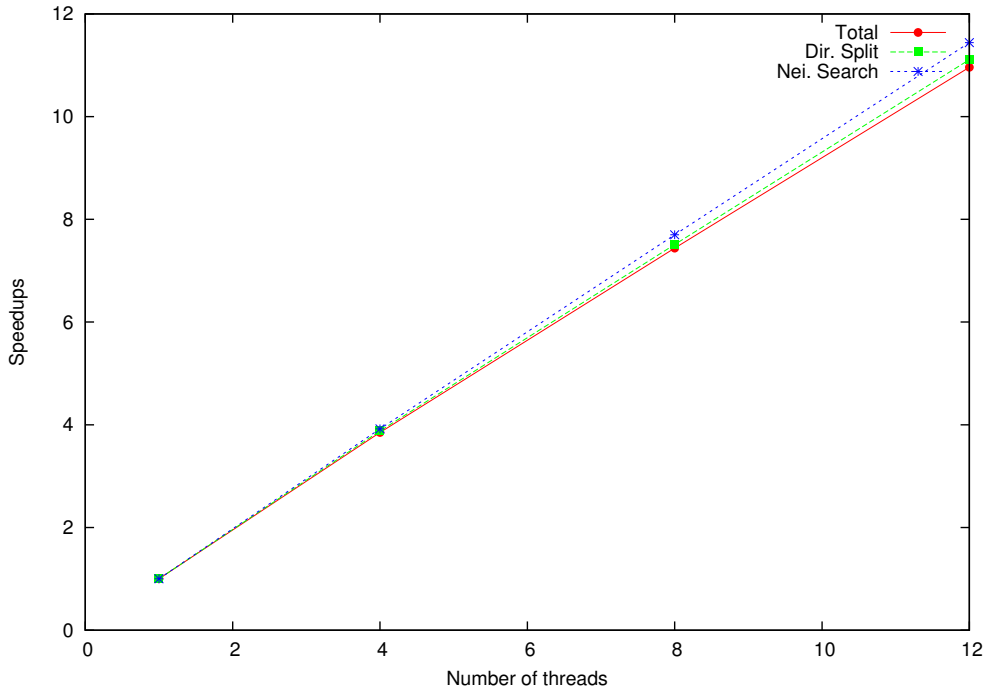


Figure 3.6: Scalability test (speedups) on the simulation of 3D gas jet into vacuum. There are 723049 fluid particles in the simulation. The dots denote the speedups for the entire time step, the squares denote the speedups in the directional splitting routine, and the stars denote the speedups in the neighbour search routine.

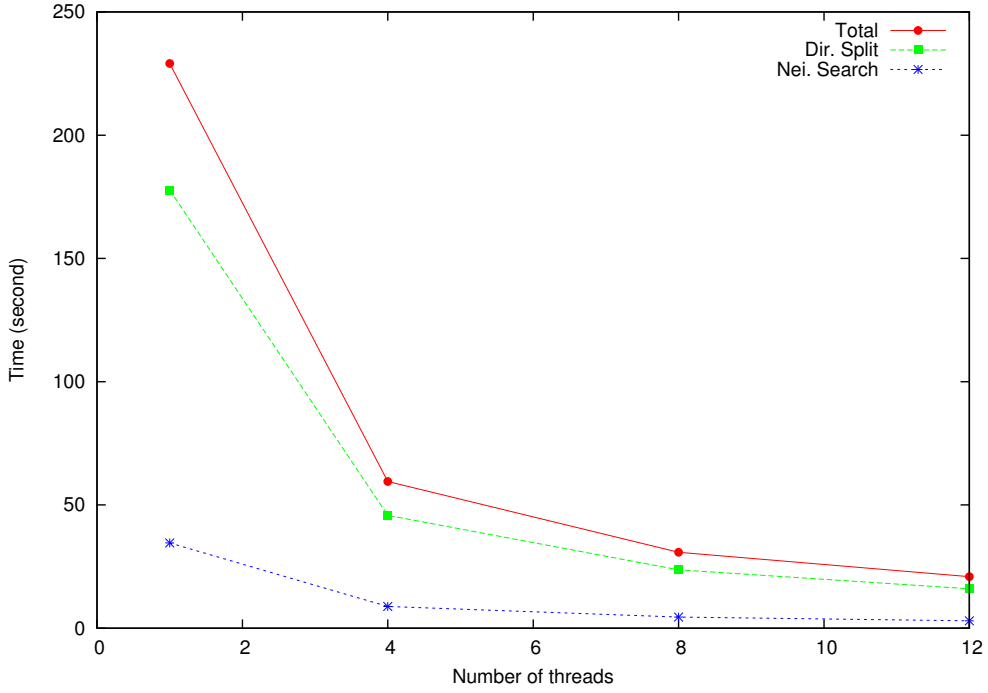


Figure 3.7: Simulation time of one time step in the simulation of 3D gas jet into vacuum. There are 723049 fluid particles in the simulation. The dots denote simulation time of the entire time step, the squares denote the time spent on the directional splitting routine, and the stars denote the time spent on the neighbour search routine.

3.4.1 1D Gaussian Pressure Wave Propagation with Periodic Boundaries

We study the propagation of a pressure wave in gas with the constant initial density $\rho = 0.01$ and the initial Gaussian pressure distribution

$$p = 5 + 2e^{-100x^2} \tag{3.82}$$

in the domain $-1.5 \leq x \leq 1.5$ with periodic boundaries on both ends. The polytropic gas EOS is used with $\gamma = 5/3$. The goal of the simulation is to demonstrate the accuracy of the proposed algorithm in resolving nonlinear waves with the formation of shocks. The benchmark data is obtained using a highly refined, grid-based MUSCL scheme. The results, shown in Figure 3.8, are labeled as *1st* for the first order local polynomial fitting, *B.W.* for the Beam-Warming scheme with second order local polynomial fitting, and *B.W. lim.* for the Beam-Warming scheme with the second order local polynomial fitting with limiter, respectively. As expected, first order scheme is diffusive, while the Beam-Warming scheme is dispersive near discontinuities. However, results demonstrates that the proposed limiter method effectively reduces dispersions near sharp edges, resulting in maintaining globally the second order of convergence.

We have also verified that the Lagrangian particle methods accurately resolves waves in stiff materials. Using the same initial conditions as before but

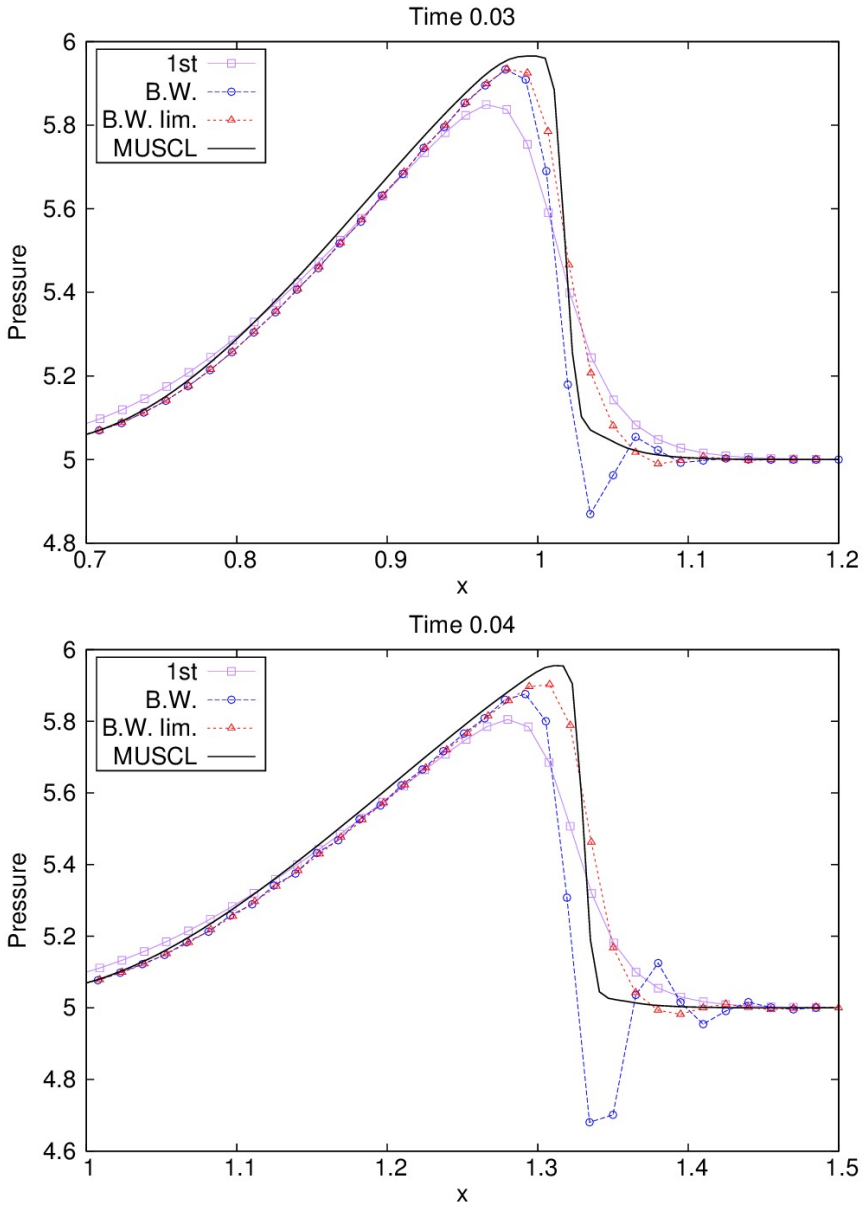


Figure 3.8: Gaussian pressure wave propagation with periodic boundaries at time 0.03 (top) and 0.04 (bottom). Coarse-resolution simulations results were used to illustrate the behavior qualitatively.

Number of particles	Relative $L2$-norm error	Rate of Convergence
240	0.051	NA
480	0.018	2.88
960	0.0049	3.60
1920	0.0012	4.02
3840	0.00029	4.23
7680	0.000068	4.24

Table 3.1: Convergence for the polytropic gas EOS case with $\gamma = \frac{5}{3}$ and initial density $\rho_0 = \frac{1}{V} = 0.01$

Number of particles	Relative $L2$-norm error	Rate of convergence
240	0.069	NA
480	0.021	3.23
960	0.0056	3.84
1920	0.0014	3.97
3840	0.00035	4.0
7680	0.000093	3.8

Table 3.2: Convergence for the stiffened polytropic gas EOS case with $\gamma = 6$, Pinf $= 7000$, einf $= 0$, and initial density $\rho_0 = \frac{1}{V} = 1$

replacing the polytropic EOS with the stiffened polytropic EOS

$$E = \frac{(P + \gamma P_\infty) V}{\gamma - 1} \tag{3.83}$$

with $\gamma = 6$ and $P_\infty = 7000$, we . The convergence results can be found in Tables 3.1 and 3.2. In both cases, second order convergence is obtained.

3.4.2 2D Gaussian Pressure Wave Propagation with Free Surface

To test the proposed algorithm for two-dimensional problems involving free surfaces, a circular disk of particles with stiffened polytropic gas EOS (with $\gamma = 6$, pinf = 7000, and einf = 0, $\rho = 1$) and a Gaussian pressure profile was initialized. The results are presented in two dimensions in Figure 3.9. Note that the latest-time plot in Figure 3.9 represents the state when the pressure waves have been reflected from the free surface for more than ten times. To verify the accuracy, the analogous one-dimensional problem with cylindrical coordinates under the Eulerian formulation is solved using a refined MUSCL scheme with the method of front tracking for the free surface implemented in the FronTier code [11]. The location and shape of the pressure wave and the interface as well as the oscillatory motion of the free surface are in good agreement with the Frontier simulation. The verification test and the fact that the pressure wave maintains good symmetry after many reflections from the free surface demonstrate that the method for modeling vacuum introduced in section 3.2.6 works well with the proposed algorithm.

3.4.3 2D Collision Between Two Circular Disks

In previous test problems, particles are initialized using regular distributions, such as the hexagonal packing, and slightly move with the flow. Nevertheless, the magnitude of the particle movement is quite restricted in previous tests, usually less than five percent of the initial inter-particle-spacing. In this sec-

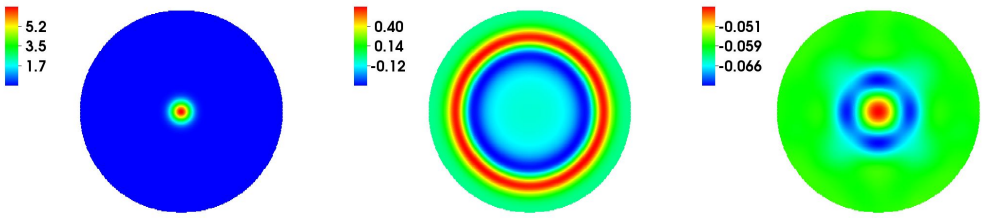


Figure 3.9: 2D Gaussian pressure wave propagation in disk with free surface. Pressure distribution (bar) at initial time (left), 10 (middle), and 60 (right).

tion, a geometrically complex two-dimensional problem with intense particle movement and object shape distortion is discussed.

The setup of the problem is as follows. Two fluid disks have initially uniform density $\rho = 1$ and zero pressure, and material properties described by the stiffened polytropic EOS with $\gamma = 6$ and $P_\infty = 7000$. The two disks move toward each other with the relative longitudinal velocity of 20, but along lines that do not connect their centers. The time sequence shows the distortion of disks after the collision. While no benchmark data exists for such a problem, we believe that the results are reasonable from physics point of view as they agree with theoretical estimates of achievable pressure peaks. They demonstrate the ability of the proposed method to handle geometrically complex interfaces.

3.4.4 2D and 3D PJMIF related simulations

In this section we show 2D and 3D simulation results of free-surface problems using the proposed Lagrangian particle method. The free-surface problems of

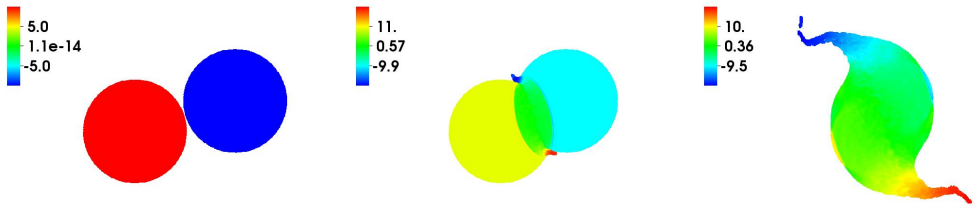


Figure 3.10: 2D simulation of collision of two disks. Velocity distribution (10 m/s) at initial time (left), 21 (middle), and 54 (right).

interest are related to the Plasma Jet driven Magneto-Inertial Fusion (PJMIF).

PJMIF is a new fusion concept that serves as a potential propulsion technolgy for interstellar missions. PJMIF uses converging plasma jets that form a uniform liner, which compresses a magnetized target to fusion conditions. It is a hybrid method of the Inertial Confinement Fusion (ICF) and Magnetic Confinement Fusion (MCF), featuring potential benefits such as lower system mass and lower cost. [38] suggest that a plasma liner can be formed by the merger of a large number of radial, highly supersonic plasma jets. These plasma jets implode on a magnetized plasma target and compresses it to conditions of the fusion ignition. Such a plasma liner is assembled when the jets intersect and merge with each other at the intermediate radius r_m, as shown schematically in Figure 3.11. For recent works on the dynamics of spherically symmetric liners imploding on deuterium plasma targets or undergoing the self-implosion process and properties of plasma liner/target after the implosion, one can refer to [40–43]. The Plasma Liner Experiment (PLX) group in Los Alamos National Lab (LANL) proposed a collaborative project to explore the feasibility of plasma liner formation to reach a desirable stagnation pressure. They fo-

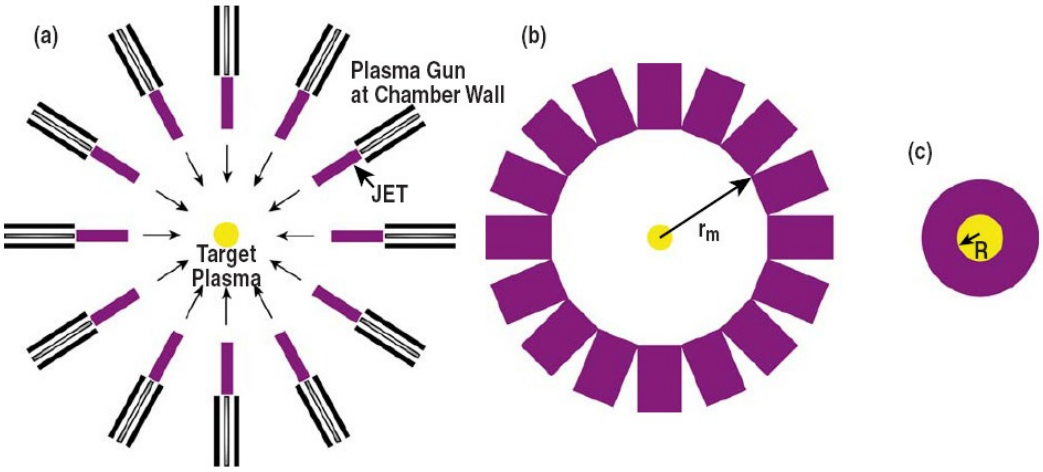


Figure 3.11: Schematic of plasma jet induced magnetized target fusion: (a) Plasma gun shoot supersonic jets; (b) Plasma liner is formed at the merging radius; (c) Plasma liner implodes on the target. Plot courtesy: [39]

cused on the plasma liner magneto-inertial fusion and analysed the efficiency of conversion [44] and preformed one-dimensional radiation-dyfrodynamic simulations to show insights into the scaling of stagnation pressure. They also performed three-dimensional SPH simulations to explore on the effects of discrete plasma jets on the processes of plasma liner formation and self-implosion [42]. Recently, using the pulse-power-driven plasma railgun produced by the HyperV Technologies Corp, results of single argon plasma jet propagation [41] and two argon plasma jets merger [43] were reported by the PLX group.

In the next subsections, we first present two and three-dimensional simulations results of a single gas jet expansion into vacuum using the Lagrangian particle method. Then we analyze results of previous three-dimensional simulations of the jet merger and the liner formation and implosion done in [48] using the grid-based MUSCL solver of the Frontier code and describe the cur-

rent work in progress based on the Lagrangian particle code.

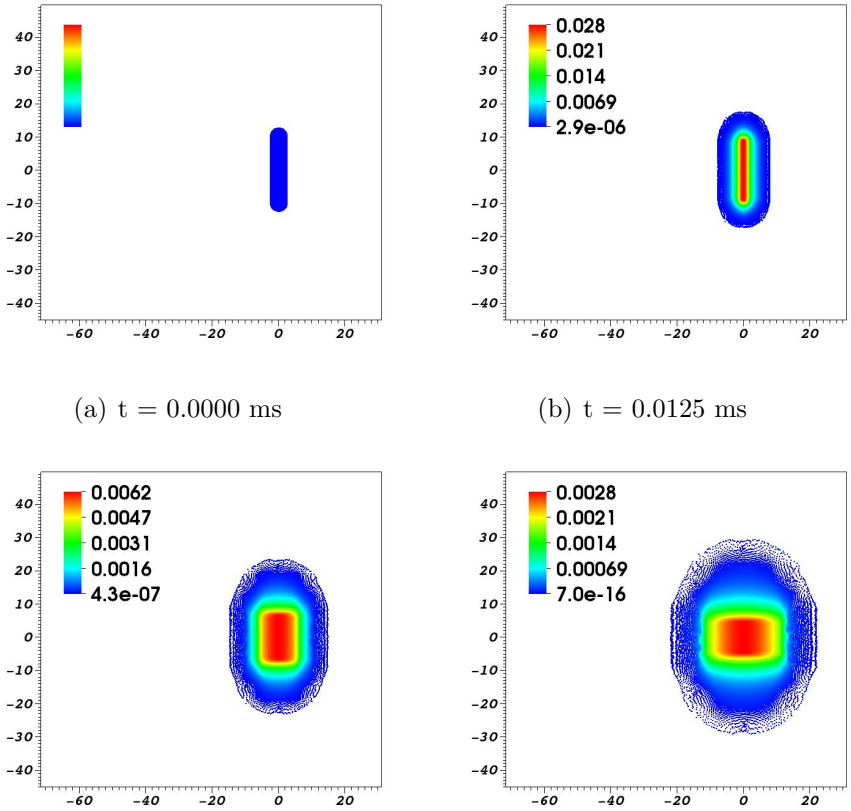
2D and 3D Simulations of Jet Expansion Into Vacuum

We use the Lagrangian particle code to solve the single gas jet expansion into vacuum problem with the same parameters as specified in [41]. To be specific, the states of are: $\rho = 1.327\text{e}{-6}$ (g/cm^3), temparature $T = 1.4$ (eV), and velocity $V = 30$ (km/s). In 2D, the jet consists of a rectangle of side length 5 (cm) and 20 (cm), and two semi-circles with radius 2.5 (cm) on both ends of the rectangle. The equation of state is the polytropic gas EOS with $\gamma = 5/3$. Figure 3.12 show the evolution of pressure of the 2D jet when it expands into vacuum. The physical time is from time 0 to 0.0125 ms. The numerical resolution, which is represented by the initial spacing among particles, is 0.05 (cm). There are 55489 fluid particles at initialzation.

In 3D, the jet consists of a cylinder with radius 2.5 (cm) and side length 10 (cm), and two semi-spheres with radius 2.5 (cm). The states are the same as the 2D case. Figure 3.13 shows the evolution of pressure of the 3D jet when it expands into vacuum. The physical time is from time 0 to 0.01 ms. The numerical resolution, which is represented by the initial spacing among particles, is 0.1 (cm). There are about 507358 fluid particles at initialzation and the simulations was run using 24 threads.

2D and 3D Simulations of Two Jet Merger

[48] reports results of three-dimensional simulations of the jet merger and the liner formation and implosion. They examine the structure of the liner



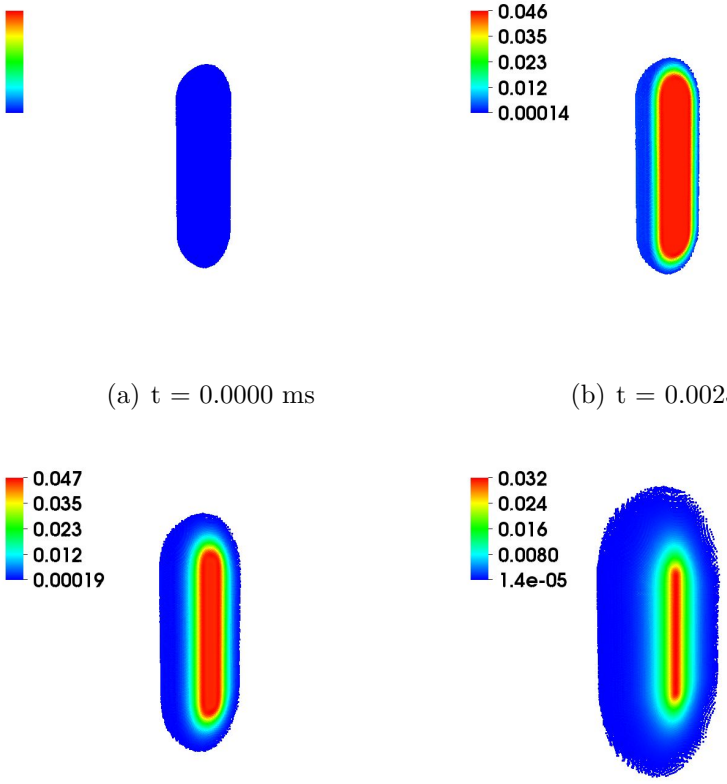
(a) t = 0.0000 ms

(b) t = 0.0125 ms

(c) t = 0.0250 ms

(d) t = 0.0375 ms

Figure 3.12: Pressure profile of the 2D jet expansion into vacuum. Figure 3.12(a) refers to the initial pressure, which is 4.873e−2 (bar). Figure 3.12(b) refers to the pressure before the merging radius.



(a) t = 0.0000 ms (b) t = 0.0025 ms

(c) t = 0.0050 ms (d) t = 0.0100 ms

Figure 3.13: Pressure profile of the 3D jet expansion into vacuum. Figure 3.13(a) refers to the initial pressure, which is 4.873e−2 (bar).

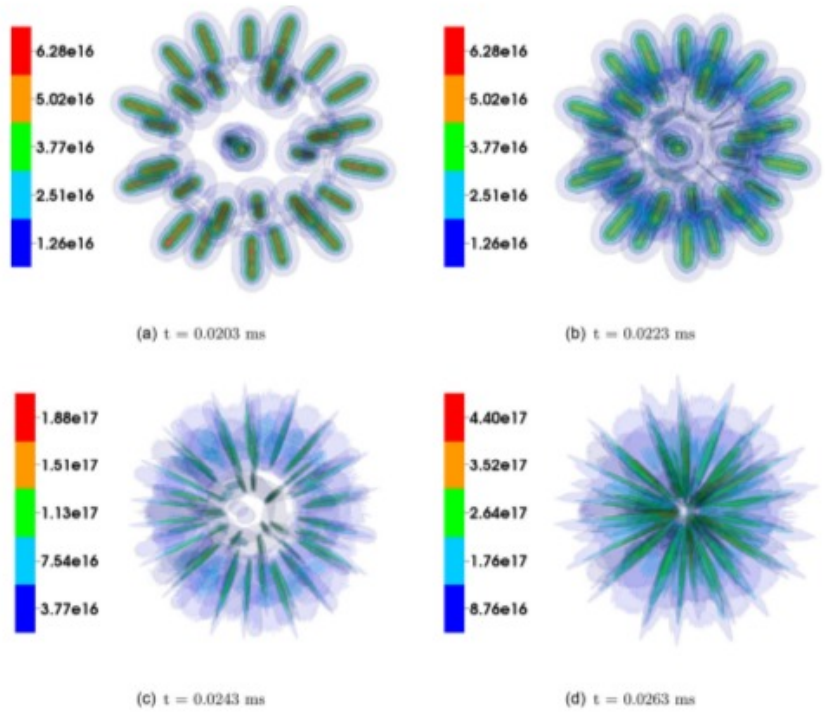


Figure 3.14: Density ($1/cm^3$) contours before merger (a, b) and after merger (c, d) of 30 argon plasma jets. Plot courtesy: [48].

obtained by the merger of 30 plasma jets, the liner uniformity, the reduction of the Mach number during the liner implosion, and compare with the theory of oblique shock waves. Specifically, oblique shock waves heat the liner, reduce the average Mach number, and contribute to the liner non-uniformity. Figures 3.14 and 3.15 show the density and pressure before and after merger of these 30 jets. During the jet merger process, the highest pressure appears along the plane of interaction of the neighboring jets and the formation of high pressure contours having shapes of the pentagon and hexagon are observed. They also perform simplified two-dimensional simulations and obtained useful information on the internal structure of plasma liners.

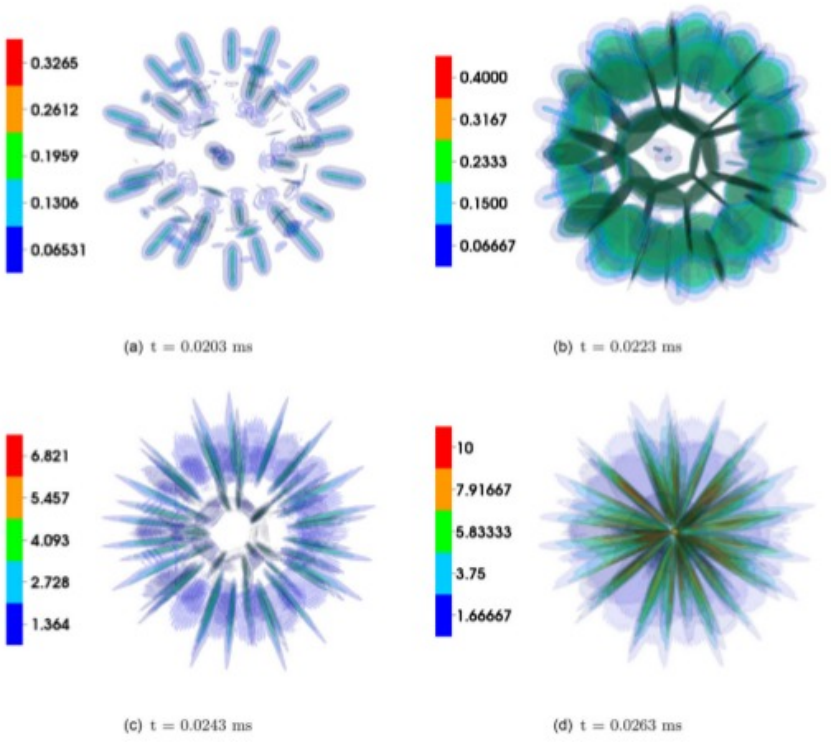


Figure 3.15: Pressure (bar) contours before merger (a, b) and after merger (c, d) of 30 argon plasma jets. Plot courtesy: [48].

In [48], the MUSCL solver within the FronTier code was used to carry out the 2D and 3D jet merger simulations. FronTier is a hybrid LagrangianEulerian code based on the front tracking method [49]. The ability to propagate and resolve topological changes of dynamically moving fronts (interfaces) between materials using Lagrangian meshes, is an important and special property of this code. Wave simulations rely mostly on FronTiers interior solvers that implement shock capturing schemes such as the Monotonic Upstream-centered Scheme for Conservation Laws (MUSCL) and WENO. The FronTier code has been used on various supercomputers for the simulation of fundamental (turbulent fluid mixing) and applied problems (liquid accelerator targets, fuel jets, pellet fueling of tokamaks, etc.).

However, the FronTier code does not support adaptive mesh refinements, critically important to this problem. As a result, mesh refinements needs to be done manually by stopping the simulation, saving the results, and restart using a finer resolution. The Lagrangian particle method is perfectly suited for PJMIF simulations because of its natural adaptivity and ability to simulate extremely non-uniform domains. To enable simulations of the plasma jet merger and plasma liner formation, we have started work on (a) implementation of EOS with support of atomic physics processes in high-Z materials, based on the Zeldovich average ionization model in local thermodynamic equilibrium [50], (b) radiation model in thin optical limit based on accurately pre-computed opacity values, and (c) improvement of the ability of the code to work in strongly compressible regime. In section 3.4.3 we have shown that the Lagrangian particle method works fine for free surface problems in the low

compressibility fluid regime. We have started the work on highly compressible regime, characterized by orders of magnitude density changes, which requires additional development.

Chapter 4

Conclusions and Future Work

In this thesis we described the fundamentals of the SPH method for the simulation of hydrodynamics , elaborated on the enhancements with new implementation of physics models (cavitation, boundary con- ditions etc.), presented the simulations of mercury targets interacting with strong proton pulses and the mercury thimble experiments, and explained in details why the SPH method produces stable and reasonable results for certain problems. To recall, the traditional discrete SPH equations for the compressible Euler equations are not accurate, but they accurately represent equations of the Lagrangian dynamics of particles interacting via isentropic potentials. The problem is that the Hamiltonian dynamics of particles only approximately represent the dynamics of continuum hydrodynamic systems. Furthermore, these traditional SPH discretization only has zeroth-order convergence for widely used kernels, and corrected or modern SPH methods such as the moving-least-squares SPH, Godunov-SPH, P-SPH, PHANTOM etc, improve on inaccurate SPH deriva-

tives, yet at the expense of of other features such as conservation, long-time stability, or prohibitively large number of neighbors that may lead to other problems.

While SPH has disadvantages, it has many good features as a candidate for simulations of the free surface or multiphase problems with complex geoemtries. Motivated by the need to resolve SPH failures while preserving its advantages, a Lagrangian particle method has been proposed for the simulation of Euler equations describing compressible inviscid fluids or gases. By representing Lagrangian fluid cells with particles, similarly to SPH, the method eliminates the mesh distortion problem of the original Lagrangian method and is suitable for the simulation of complex free surface flows. The main contributions of our method, which is different from SPH in all other aspects, are (a) significant improvement of approximation of differential operators based on polynomial fits and the corresponding least squares problems and convergence of prescribed order, (b) an upwinding second-order particle-based algorithm with limiter, providing accuracy and long term stability, and (c) accurate resolution of states at free surfaces based on the technique of ghost particles. Numerical verification tests demonstrate the second convergence order of the method and its ability to resolve complex free surface flows.

The Lagrangian particle method has numerous advantages compared to grid-based methods for the simulation of complex systems. It eliminates the need for complex and costly algorithms for the generation and adaptation of meshes, provides continous adaptivity to density changes, and is suitable for extremely non-uniform domains typical for astrophysics or high energy

density applications. The algorithmic complexity of key particle methods insignificantly increases with the increase of spatial dimensions, making a 3D code similar to a 1D code. In addition, particle algorithms are independent of the geometric complexity of domains. In contrast, there is a huge increase in algorithmic complexity of a 3D mesh generation and dynamic adaptation compared to 1D as well as the increase associated with the geometric complexity of domains.

The future development of the space-time discretization methods will explore new high resolution WENO-type solvers based on irregularly placed particle nodes and symplectic integrators. In addition, we will also work on improvements of the ability of the code for strongly compressible regime. In order to run the plasma jet merger and plasma liner formation, future work will also include the implementation of EOS with support of atomic physics processes in high-Z materials, based on the Zeldovich average ionization model in local thermodynamic equilibrium, and the radiation model in thin optical limit based on accurately pre-computed opacity values. Our Lagrangian particle method is also generalizable to coupled multiphysics systems, including the dynamics of incompressible fluids, solids, and plasmas.

Bibliography

[1] H. Chen, R. Samulyak, W. Li, *Lagrangian Particle Method for Compressible Fluid Dynamics*, J. Comput. Phys., 2015, submitted.

[2] , R. Samulyak, H. Chen, H. Kirk, K. McDonald, *Simulation of high-power mercury jet targets for neutrino factory and muon collider*, Proc. NA-PAC 2013, North American Particle Accelerator Conference, Sep. 29 0 Oct. 3, 2013, Pacadena, CA, paper TUPBA09

[3] , H. Chen and R. Samulyak, *Lagrangian Particle Method for Compressible Euler Equations based on Generalized Finite Differences*, 12th U.S. National Congress on Computational Mechanics, Raleigh, North Carolina, July 22-25, 2013.

[4] , X. Wang, R. Samulyak, X. Jiao, K. Yu, *Optimal Generalized Finite Difference Solution to Particle-in-Cell Problem*, IPAC-2015, 6th International Particle Accelerator Conference, May 3-8, 2015, Richmond, VA. Paper MOPMN040.

[5] , X. Wang, R. Samulyak, J. Jiao, K. Yu, Adaptive Particle-in-Cloud Method for Optimal Solutions to Vlasov-Poisson Equation, J. Comput. Phys., 2015. Submitted

[6] R.D. Richtmyer, K.W. Morton, *Finite diference methods for initial value problems*, Interscience, New York - London - Sydney, 1967.

[7] H. Lamb, *Hydrodynamics*, Cambridge Univ. Press,

[8] Hirt, C.W., Nichols, B.D. (1981), "Volume of fluid (VOF) method for the dynamics of free boundaries", J. Comput. Phys., 39 (1): 201225

[9] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, 2002, Springer-Verlag.

[10] C.W. Hirt, A.A. Amsden, J.L. Cook, An arbitrary Lagrangian-Eulerian computing method for a;; flow speeds, J. Comput. Phys, 135 (1997), 203-216.

[11] B. Fix, J. Glimm, X. Li, Y. Li, X. Liu, R. Samulyak, and Z. Xu. A TSTT integrated Frontier Code and its applications in computational fluid physics. *Journal of Physics: Conf. Series*, 16:471–475, 2005.

[12] L. B. Lucy, A numerical Approach to The Testing of THe Fission Hypothesis, *The Astronomical Journal*, Vol. 82, Dec. 1977, p. 1013 - 1024

[13] R. A. Gingold, J. J. Monaghan, Smoothed Particle Hydrodynamics: Theory and Applications To Non-Spherical Stars, *Mon. Not. R. Astro. Soc.*, (1977) **181**, 375-389

[14] J. J. Monaghan, Smoothed Particle Hydrodynamics, In: Annual review of astronomy and astrophysics. Vol. 30, p. 543-574. 1992.

[15] J. J. Monaghan, Smoothed particle hydrodynamics, Rep. Prog. Phys., 68 (2005), 1703 – 1759.

[16] J. J. Monaghan, Smoothed particle hydrodynamics and Its Diverse Applications, Annual Review of Fluid Mechanics, Vol. 44, 323-346 (2012).

[17] J. J. Monaghan, J. C. Lattanzio. A Refined Particle Method for Astrophysical Problems, Astronomy and Astrophysics, vol. 149, no. 1, Aug. 1985, p. 135-143.

[18] J. J. Monaghan, On the Problem of Penetration in Particle Methods, Journal of Computational Physics, **82**, 1-15, 1989

[19] J. J. Monaghan, Simulating Free Surface Flows with SPH, *Journal of Computational Physics*, **110**, 399 - 406 (1994)

[20] G. A. Dilts, Moving-least-squares particle hydrodynamics I. Consistency and stability. Int. J. Num. Methods in Engineering, 44 (1999), 1115 – 1155.

[21] P. F. Hopkins, GIZMO: a new class of accurate, mesh-free hydrodynamic simulation methods, Mon. Not, R. Astron. Soc., 2014.

[22] D. Price, Smoothed particle hydrodynamics and magnetohydrodynamics, arxiv.org:1012.1885v1

[23] J.L. Steger, R.F. Warming, Flux vector splitting of the inviscid gas dynamic equations with application to finite-difference methods, J. Comput. Phys., 40 (1981), 263 - 293.

[24] G. Strang, On the construction and comparison of difference schemes, SIAM J. Numerical Analysis, 5 (1968), 506-517.

[25] J.J. Benito, F. Urena, L. Gavete, Influence of several factors in the generalized finite difference method, Applied Math. Modeling, 25 (2001), 1039-1053.

[26] R. Courant, K.O. Friedrichs, *Supersonic flow and shock waves*, Springer, 1999.

[27] Jianqing Fan, Theo Gasser, Irene Gijbels, Michael Brockmann, and Joachim Engel. Local polynomial fitting: A standard for nonparametric regression, 1993.

[28] X. Jiao and H. Zha. Consistent computation of first- and second-order differential quantities for surface meshes. *ACM Solid and Physical Modeling Symposium*, 2008. (Available at arxiv.org/abs/0803.2331)

[29] G. GOLUB and C. F. VAN LOAN. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 1996.

[30] H. Samet, The design and analysis of spatial data structures, Adison-Wesley Publishing Company, 1990.

[31] Chi-Wang Shu, Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws, NASA/CR-97-206253, ICASE Report No. 97-65, Nov. 1997

[32] G. R. Liu, M. B. Liu, Smoothed Particle Hydrodynamics: A Meshfree Particle Method, 2003

[33] R. A. Darlrymple, O. Knio, SPH Modelling of Water Waves, *Proc. Coastal Dynamics*, Lund, 2001

[34] L. D. Libersky and A. G. Petscheck (1991), Smoothed particle hydrodynamics with strength of materials, in H. Trease, J. Fritts and W. Crowley (ed.): Proceedings of The Next Free Lagrange Conference, Springer-Verlag, NY, 395:248-257

[35] L. Verlet, Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Phys. Rev.*, 1967, 159: 98 - 103

[36] B. J. Leimkuhler, S. Reich, R. D. Skeel, Integration Methods for Molecular dynamic IMA Volume in Mathematics and its application. Springer 1997

[37] A. Fabich, CERN-THESIS-2002-038, Vienna, November (2002).

[38] Y. C. F. Thio, E. Panarella, C. E. Knupp R. C. Kirkpatrick, F. Wysocki, P. Parks, and G. Schmidt. Magnetized target fusion in a spheroidal geometry with standoff drivers. in Current Trends in International Fusion Research II, edited by E. Panarella, National Research Council Canada, Ottawa, Canada, 1999.

[39] P. B. Parks. On the efficacy of imploding plasma liners for magnetized fusion target compression. Phys. Plasmas, 15:062506, 2008.

[40] R. Samulyak, P. Parks, and L. Wu. Spherically symmetric simulation of plasma liner driven magnetoinertial fusion. Phys. Plasmas, 17:092702, 2010.

[41] S. C. Hsu, E. C. Merritt, A. L. Moser, T. J. Awe, S. J. E. Brockington, J. S. Davis, C. S. Adams, A. Case, J. T. Cassibry, J. P. Dunn, M. A. Gilmore, A. G. Lynn, S. J. Messer, and F. D. Witherspoon. Experimental characterization of railgun-driven supersonic plasma jets motivated by high energy density physics applications. Phys. Plasmas, 19:123514, 2012.

[42] J. T. Cassibry, M. Stanic, S. C. Hsu, S. I. Abarzhi, F. D. Witherspoon. Tendency of spherically imploding plasma liners formed by merging plasma jets to evolve toward spherical symmetry. Phys. Plasmas, 19:052702, 2012.

[43] E. C. Merritt, A. L. Moser, S. C. Hsu, J. Loverich, M. Gilmore. Experimental characterization of the Stagnation Layer between Two Obliquely Merging Supersonic Plasma Jets. Phys. Rev. Lett, 111:085003, 2013.

[44] S. C. Hsu, T. J. Awe, S. Brockington, A. Case, J. T. Cassibry, G. Kagan, S. J. Messer, M. Stanic, X. Tang, D. R. Welch, and F. D. Witherspoon. Spherically imploding plasma liners as a standoff driver for magnetoinertial fusion. IEEE Transactions on Plasma Science, 40:1287, 2012.

[45] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational Geometry Algorithms and Applications, second edition, Springer

[46] K. Zhou, M. Gong, X. Huang, and B. Guo. Data-Parallel Octrees for Surface Reconstruction, IEEE Transactions on Visulization and Computer Graphics

[47] Van Leer, B. Towards the ultimate conservative difference scheme II. Monotonicity and conservation combined in a second order scheme, J. Comp. Phys. 14 (4): 361370, 1974

[48] H. Kim, L. Zhang, R. Samulyak, and P. Parks, On the structure of plasma liners for plasma jet induced magnetoinertial fusion, Phys. Plasmas 20, 022704 (2013); doi: 10.1063/1.4789887

[49] J. Glimm, J. Grove, X. L. Li, K. L. Shyue, Q. Zhang, and Y. Zeng, Three dimensional front tracking, SIAM J. Sci. Comput. 19, 703727 (1998).

[50] Ya. B. Zeldovich and Yu. P. Raizer, Physics of Shock Waves and High-Temperature Hydrodynamic Phenomena (Dover, 2002).

[51] T. Guo, PhD thesis, Department of Applied Mathematics and Statistics, Stony Brook University, 2013.