Stony Brook University

Software Tools for Stochastic Simulations of Turbulence

A Dissertation Presented

by

Ryan Kaufman

to

The Graduate School

In Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Computational Applied Mathematics

Stony Brook University

December 2014

Stony Brook University

The Graduate School

Ryan Kaufman

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

James Glimm - Dissertation Advisor

Professor, Department of Applied Mathematics and Stastics

Xiaolin Li - Chairperson of Defense

Professor, Department of Applied Mathematics and Stastics

Roman Samulyak - Member

Professor, Department of Applied Mathematics and Stastics

Foluso Ladeinde - Outside Member

Associate Professor, Department of Mechanical Engineering

This dissertation is accepted by the Graduate School

Charles Taber

Dean of the Graduate School

Abstract of the Dissertation

Software Tools for Stochastic Simulations of Turbulence

by

Ryan Kaufman

Doctor of Philosophy

in

Applied Mathematics and Statistics

Computational Applied Mathematics

Stony Brook University

2014

We present two software tools useful for the analysis of mesh based physics application data, and specifically for turbulent mixing simulations. Each has a broader, but separate scope, as we describe. Both features play a key role as we push computational science to its limits and thus the present work contributes to the frontier of research.

The first tool is Wstar, a weak* comparison tool, which addresses the stochastic nature of turbulent flow. The goal is to compare underresolved turbulent data in convergence, parameter dependence, or validation studies. This is achieved by separating space-time data from state data (e.g. density, pressure, momentum, etc.) through coarsening and sampling. The collection of fine grained data in a single coarse cell is treated as a random sample in state space, whose cumulative distribution function defines a measure within that cell. This set of measures with the spacial dependence defined by the coarse grid defines a Young measure solution to the PDE.

The second tool is a front tracking application programming interface (API) called FTI. It has the capability to generate geometric surfaces (e.g. the location of interspecies boundaries) of high complexity, and track them dynamically. FTI also includes the ghost fluid method, which enables mesh based fluid codes to maintain sharpness at interspecies boundaries by modifying solution stencils that cross such a boundary. FTI outlines and standardizes the methods involved in this model. FronTier, as developed here, is a software package which implements this standard. The client must implement the physics and grid interpolation routines outlined in the client interface to FTI. Specific client programs using this interface include the weather forecasting code WRF; the high energy physics code, FLASH; and two locally constructed fluid codes, cFluid and iFluid for compressible and incompressible flow respectively.

Dedicated to my family and friends. without whose support, I would not be here today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The motivation for this thesis is to provide simple and useful software tools to facilitate turbulent mixing simulations. We introduce two tools, each of which took on a life of its own, each with a distinct generalization beyond its original target domain. As a consequence, we consider them separately, as distinct sections of the thesis. Though developed as techniques for Navier-Stokes based turbulent LES, we also discuss the broader range of possible applications for each.

It has been shown that front tracking with large eddy simulations (LES) gives good agreement with experimental data for instability studies, such as Rayleigh-Taylor (RT) [15, 20]. Convergence is challenging to establish in an unstable regime [21]. For this we introduce a stochastic notion of convergence, named weak$*$ convergence (or $w*$ convergence). Weak* convergence is a very general concept. In this thesis, we consider weak* convergence of probability measures. In this weak* limiting process, we realize the solution of a PDE as a space-time dependent probability measure. This form of stochastic convergence has been developed in [37, 38, 7, 4] and used by [10] in their existence proof for 3D Euler equations.

We outline techniques for computing Navier-Stokes solutions based upon

- LES / SGS
- Front Tracking
- Stochastic Convergence

Chapter 3 deals with our first main theme: stochastic data analysis. It is natural to consider turbulent flows as a stochastic process. We present a portable tool which can generate stochastic data from a single deterministic simulation output. We calculate various statistics, including arbitrary order single point correlations between physical variables, as well as probability density functions (PDFs) and cumulative density functions (CDFs) for

state-space variables. We show verification studies for the tool as well. The unique contribution of this thesis is the implementation of this tool, as well as participation in verification studies based on its use.

Chapter 4 introduces our second theme: bringing a software innovation, front tracking, into ready use in a wide range of computational simulation codes. Front Tracking has been verified and validated in RT [31, 12, 11, 24, 26] studies using SGS in an LES regime. The unique contribution of this thesis is the design of the API based on past implementations, and the implementation of the server code, as well as the client code in cFluid, FLASH, and WRF.

Chapter 2

Introduction to Turbulence Models

2.1 Turbulence Modeling

To discuss turbulence and its length scales, we introduce the dimensionless Reynolds number, which is a ratio of the inertial to the viscous forces in a fluid flow. The Reynolds number is defined as

$$Re = \frac{\text{Inertial Force}}{\text{Viscous Force}} = \frac{\rho v L}{\mu} = \frac{v L}{\nu}\,, \tag{2.1}$$

where L and v are typical length and velocity values, μ and ν the dynamic and kinematic viscosities. Since viscosity is the mechanism by which small turbulent vortices break down, a high Reynolds number implies finer vortex structures associated with turbulence. The range of turbulent length scales is divided into three main range, each with a preferred computational method.

- Integral length scales
- Inertial length scales
- Dissipation length scales

The integral range is the range of energy scales associated with the largest eddies in the flow structure. The size of these large eddies is bounded only by the size of the flow medium. They contain a large amount of energy in low frequency oscillations. Integral length scale vortices supply the energy which drives the smaller scale vortices.

The inertial range is an intermediate range. Vortices in this range are in balance between the driving forces of the large scale eddies and the dissipative forces of the small scale eddies. Scales in this range are non-dissipative, however the dissipation rate at smaller scales controls the rate of energy transfer in this range. Since vortices on each scale are

experiencing balanced energy transfer, there is self similarity in in this range [23] and the energy can be expressed as a function of only two parameters: κ, the wave number; and ϵ, the dissipation rate. In this range, the slope of the log of energy as a function of the log of the wave number, $\ln(\kappa)$ vs $\ln(E(\kappa))$ (for fixed ϵ) remains constant and is represented by the Kolmogorov $\frac{5}{3}$ power law [23].

The dissipation range is the range of length scales where the viscous effects of the medium are non-negligable. This range begins with the Kolmogorov scale η, the length of the smallest vortices, defined as:

$$\eta = \left(\frac{\nu_k^3}{\epsilon}\right)^{1/4}, \tag{2.2}$$

where ν_k is the kinematic viscosity. Below the Kolmogorov scale, turbulent vortices cannot exist due to being overdamped by viscous forces. Turbulent eddies in this range have high frequency and low energy. Turbulent effects do not extend beyond this range.

We classify techniques for computing solutions to Navier-Stokes equations into three distinct categories:

1. DNS: Direct Numerical Simulation
2. LES: Large Eddy Simulation
3. RANS: Reynolds Averaged Navier-Stokes

Direct numerical simulation (DNS) is the most fundamental solution technique. It computes the PDE solution directly, i.e. with no additional models for turbulence. DNS describes these solutions in which flow structures are resolved on all length scales. This simple and ideal case is impractical for most problems involving turbulence (with models to be discussed later). Thus, it is not discussed further here.

Reynolds-averaged Navier-Stokes (RANS) models apply the Reynolds decomposition to the original PDE. This decomposition separates the solution into an averaged component, and a fluctuating component. In these averaged equations some high order terms (such as Reynolds stress) which arise from its formulation must be modeled. A RANS model treats the Reynolds stress, or more commonly, some components of it, as dynamic variables with their own equations. The method is efficient regardless of grid resolution, but is dependent on data to set the several parameters in its models. Often the coefficients for these modeled terms must be calibrated from experimental data or from LES.

2.2 Large Eddy Simulations

LES describes a technique for computing in the regime where large scale turbulent vortices are resolved, but fine scale turbulent structures still exist below the mesh resolution. LES models also begin with the filtered Navier-Stokes equations. We apply dynamic subgrid-scales (SGS) to model subgrid turbulent effects. Kolmogorov's self similarity law [23] allows for the SGS coefficients to be solved by comparing resolved effects at two different length scales in the inertial range (see §2.2.3). Therefore, the SGS models can only be applied when the characteristic length scale of the mesh is within this range. This gives a parameter free model for subgrid turbulent effects. We also apply front tracking (see Chapter 4).

The tools described in this thesis will be applicable to simulations employing LES (in the inertial regime). An expanded description appears in §2.2.

2.2.1 The Equations of Fluid Motion

The dynamics of fluid motion are modeled by Navier-Stokes equations for conservation of mass, energy, and momentum. Fundamental quantities are represented by the primitive variables $\rho, \vec{\psi}, \vec{u}, p, E$ for density, species concentration, velocity, pressure, and energy respectively. The following Navier-Stokes definitions are based on [5, 32, 28].

The equation for the conservation of mass is:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0\,. \tag{2.3}$$

The equation for the conservation of species mass in a multispecies flow is:

$$\frac{\partial \rho \psi_k}{\partial t} + \frac{\partial \rho \psi_k u_j}{\partial x_j} = -\frac{\partial (\rho \psi_k V_{k,j})}{\partial x_j}\,, \tag{2.4}$$

where $V_{k,j}$ is the diffusion velocity of species k. Using $g = \text{const}$ as the acceleration due to gravity, we have the following form for conservation of momentum:

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho g_i\,, \tag{2.5}$$

where, according to Stokes' law, we represent the viscous stress tensor, τ_{ij} by,

$$\tau_{ij} = \mu \left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right]\,. \tag{2.6}$$

with E as the total energy (per unit mass), conservation of energy is given by:

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho E u_j}{\partial x_j} + \frac{\partial p u_j}{\partial x_j} = \rho g_j u_j - \frac{\partial q_j}{\partial x_j} + \frac{\partial \tau_{ij} u_i}{\partial x_j} \tag{2.7}$$

2.2.2 Navier Stokes Filtering

The LES filter is a mathematical operation which removes small scales from the solution to the Navier-Stokes equations. It is a low-pass filter, which means that low frequency, large wavelength perturbations, i.e. large eddies, are allowed to pass, while high frequency perturbations, i.e. small eddies are filtered out.

When applying a LES model, the original Navier-Stokes equations are filtered. A filter has a length scale (denoted $\bar{\Delta}$) associated with it. $\bar{\Delta}$ defines the wavelength above which information is allowed to pass. This is called the filter width. The quantity, q, filtered by $\bar{\Delta}$, is denoted $\bar{q}$. Thus, we have,

$$q = \bar{q} + q' = \text{filtered component} + \text{subfilter component}\,. \tag{2.8}$$

The filter is defined by a convolution, $G_{\bar{\Delta}}$, associated with the length scale $\bar{\Delta}$.

We define the filtered quantity $\bar{q}$, in terms of the function $G_{\bar{\Delta}}$

$$\bar{q} = \int_\Omega G_{\bar{\Delta}}(\vec{x} - \vec{\zeta}) q(\vec{\zeta}, t) d\vec{\zeta}\,. \tag{2.9}$$

The filter $G_{\bar{\Delta}}$ satisfies $\int_\Omega G_{\bar{\Delta}}(x - \zeta) d\zeta = 1$.

The motivation is such that the Navier-Stokes numerical solution at the grid level represents the filtered Navier-Stokes solution. The sub-filter components (i.e. SGS component) are modeled (see §2.2.3) and added into the filtered solution. This filter could be implicit or explicit. Studies here use the implicit mesh-based filter. With an implicit filter, the filtering is performed simply by the grid resolution cutoff, and cell values are interpreted as mesh block averages. If filter width is fixed, the soution converges under mesh refinement to a solution of the filtered equations. The solution of the Navier-Stokes equation then requires a second limit as the filter length tends toward zero.

We also define the Favre filter operation, which is a mass weighted average based on the LES filter operation

$$\widetilde{q} = \frac{\overline{\rho q}}{\bar{\rho}}\,. \tag{2.10}$$

The Favre filter of the variable q will be expressed $\tilde{q}$.

We now define the filtered Navier-Stokes equations. The equation for conservation of

mass comes from applying the $\overline{\Delta}$ filter to Eq. 2.3:

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j}{\partial x_j} = 0\,. \tag{2.11}$$

By applying the filter to Eqn. 2.5, we have conservation of momentum:

$$\frac{\partial \bar{\rho}\widetilde{u}_i}{\partial t} + \frac{\partial \overline{\rho u_i u_j}}{\partial x_j} + \frac{\partial \bar{p}}{\partial x_i} = \bar{\rho} g_i + \frac{\partial \bar{\tau}_{ij}}{\partial x_j}\,, \tag{2.12}$$

where g_i is the gravitational force in direction i. The filtered viscous stress tensor is modeled as:

$$\overline{\tau}_{ij} = \overline{\nu_d}\left[\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} - \frac{2}{3}\frac{\partial \tilde{u}_k}{\partial x_k}\delta_i j\right]\,, \tag{2.13}$$

where $\overline{\nu}_d = \overline{\rho}\nu_k$ is the filtered dynamic viscosity. ν_k, the kinematic viscosity, is considered constant.

Let $R_{ij} = \overline{\rho u_i u_j} - \overline{\rho}\widetilde{u}_i\widetilde{u}_j$ be the residual stress tensor. This gives the alternate form for Eqn. 2.12:

$$\frac{\partial \overline{\rho}\widetilde{u}_i}{\partial t} + \frac{\partial \overline{\rho}\widetilde{u}_i\widetilde{u}_j}{\partial x_j} + \frac{\partial \bar{p}}{\partial x_i} = -\frac{\partial R_{ij}}{\partial x_j} + \bar{\rho} g_i + \frac{\partial \bar{\tau}_{ij}}{\partial x_j}\,. \tag{2.14}$$

$$\begin{aligned}\frac{\partial \overline{E}}{\partial t} + \frac{\partial(\overline{E}+\overline{p})\widetilde{u}_i}{\partial x_i} = \frac{\partial \overline{\tau}_{ij}\widetilde{u}_j}{\partial x_i} + \frac{\partial}{\partial x_i}(\overline{\kappa}\frac{\partial \widetilde{R}}{\partial x_i}) + \frac{\partial}{\partial x_i}\left((\widetilde{H}_h - \widetilde{H}_l)\overline{\rho}\tilde{T}\frac{\partial \widetilde{\psi}}{\partial x_i}\right) \\ + \left(\frac{1}{2}\frac{\partial \tau_{kk}\tilde{v}_i}{\partial x_i} - \frac{\partial q_i^{(H)}}{\partial x_i} - \frac{\partial q_i^{(T)}}{\partial x_i} - \frac{\partial q_i^{(V)}}{\partial x_i}\right)\,,\end{aligned} \tag{2.15}$$

where $\tilde{T}$ is the average temperature. We derive the filtered species diffusion equation from Eqn. 2.4.

$$\frac{\partial \overline{\rho}\tilde{\psi}_k}{\partial t} + \frac{\partial \overline{\rho}\tilde{\psi}_k\tilde{u}_j}{\partial x_j} = \frac{\partial}{\partial x_i}\left(\overline{\rho}\tilde{\tau}\frac{\partial \tilde{\psi}_k}{\partial x_j}\right) - \frac{\partial q_j^{(\psi)}}{\partial x_j}\,, \tag{2.16}$$

Equations 2.11-2.16 define our LES model equations for grid level effects. Four residual terms have been used here, namely, $R_{ij}, q_i^{(H)}, q_i^{(V)}, q_i^{(\psi)}$ which are expressed as

$$R_{ij} = \bar{\rho}(\widetilde{u_i u_j} - \widetilde{u}_i\widetilde{u}_j) \tag{2.17}$$

$$q_i^{(H)} = \overline{\rho}(\widetilde{c_p T u_i} - \tilde{c}_p\tilde{T}\tilde{u}_i) \tag{2.18}$$

$$q_i^{(T)} = \frac{1}{2}\overline{\rho}(\widetilde{u_k u_k u_i} - \tilde{u}_k\tilde{u}_k\tilde{u}_i) \tag{2.19}$$

$$q_i^{(V)} = \overline{\tau_{ij} u_j} - \overline{\tau}_{ij} \tilde{u}_j \tag{2.20}$$

$$q_i^{(\psi)} = \overline{\rho}(\widetilde{\psi u_i} - \tilde{\psi}\tilde{u}_i) \tag{2.21}$$

The derivation of these equations is shown out in [28], in which the subgrid term $\bar{\rho}(\widetilde{e_\infty v_i} - \tilde{e}_\infty \tilde{v}_i)$ is modeled as zero. $\widetilde{H}_h$ and $\widetilde{H}_l$ represent the partial specific enthalpy of the two species (heavy and light), defined as

$$\widetilde{H}_h = \tilde{e}_h + \frac{\overline{p}}{\overline{\rho}} \tag{2.22}$$

$$\widetilde{H}_l = \tilde{e}_l + \frac{\overline{p}}{\overline{\rho}} \tag{2.23}$$

where $\tilde{e}_h$ and $\tilde{e}_l$ are the specific internal energy of each species [28].

2.2.3 SGS Models

2.2.3.1 Smagorinsky Model

We begin SGS construction with the eddy-viscosity model [42] in which the additional turbulent stresses are modeled by adding an artificial eddy viscosity, ν_t. The approach treats dissipation of kinetic energy at subgrid scales as a form of molecular diffusion.

The SGS residual stress tensor R_{ij} can be decomposed into isotropic and anisotropic parts.

$$R_{ij} = (R_{ij} - R_{kk}\frac{\delta_{ij}}{3}) + R_{kk}\frac{\delta_{ij}}{3} = R_{ij}^a + R_{ij}^i = \text{anisotropic tensor} + \text{isotropic tensor} \tag{2.24}$$

The anisotropic part of R_{ij} is modeled based on turbulent viscosity

$$R_{ij}^a = -2\bar{\rho}\nu_t \tilde{S}_{ij}^a$$

where $S_{ij} = \frac{1}{2}(\frac{\partial \tilde{u}_i}{\partial x_j}\partial \tilde{u}_j \partial x_i)$ is the strain-rate tensor, and $S_{ij}^a = \tilde{S}_{ij} - \frac{\delta_{ij}}{3}\tilde{S_{kk}}$.

The classical Smagorinsky model for viscosity is:

$$\nu_t = (C_S \Delta)^2 |\tilde{S}| \tag{2.25}$$

where $|\tilde{S}| = \sqrt{2\tilde{S_{ij}}^2}$ and C_S is a model coefficient. Accordingly, for the anisotropic part,

$$R_{ij}^a = -2\bar{\rho}(C_s\Delta)^2 |\tilde{S}| \tilde{S}_{ij}^a \,. \tag{2.26}$$

The isotropic part is based on [45] following $k_{SGS} = (C_I\Delta)^2|\tilde{S}|^2$. We have

$$R_{kk} = 2\bar{\rho}(C_I\Delta)^|\tilde{S}|^2\,. \tag{2.27}$$

The full SGS stress tensor is modeled as

$$R_{ij} = -2\bar{\rho}(C_S\Delta)^2|\tilde{S}|\tilde{S}^a_{ij} + \frac{\delta_{ij}}{3}2\bar{\rho}(C_I\Delta)^2|\tilde{S}|^2\,. \tag{2.28}$$

The SGS mass species transport flux, gradient transport is:

$$\lambda_{kj} = -\bar{\rho}\frac{\nu_t}{Sc_t}\frac{\partial\tilde{\psi}_k}{\partial x_j}\,, \tag{2.29}$$

where Sc_t is the turbulent Schmidt number and ν_t is the Smagorinsky viscosity (see Eqn. 2.25). The equation for turbulent thermal flux is:

$$Q_j = -\bar{\rho}\frac{v_t}{Pr_t}\frac{\partial\tilde{e}}{\partial x_j} = -\rho c_v\frac{v_t}{Pr_t}\frac{\partial\tilde{T}}{\partial x_j} \tag{2.30}$$

where Pr_t is the turbulent Prandtl number.

Four turbulent transport coefficients have been introduced, namely C_s, C_I, Pr_t, and Sc_t. Using Germano's identity, as shown in the next section, we are able to close these terms.

2.2.3.2 Dynamic Smagorinsky Model

Here we demonstrate the application of the Germano identity to close subgrid terms in the multi-species filtered Navier-Stokes according to the model posed by [25] as presented in [32]. We refer to [32] for the generation of all closure terms, but here we show C_S the Smagorinsky viscosity coefficient as an example.

SGS stress at grid filter level $\bar{\Delta}$

$$R^M_{ij} = -2\bar{\rho}(C_s\bar{\Delta})^2\sqrt{2\tilde{S}_{ij}\tilde{S}_{ij}\tilde{S}^a_{ij}} + \frac{2\delta_{ij}}{3}\bar{\rho}(C_I\bar{\Delta})^2(2\tilde{S}^2_{ij}) \tag{2.31}$$

We introduce a new test filter $\hat{\bar{\Delta}}$ larger than $\bar{\Delta}$ This filter gives a new field with scales larger than the resolved field.

SGS stress st the test filter level is represented:

$$r_{ij} = \widehat{\overline{\rho u_i u_j}} - \frac{\widehat{\overline{\rho u_i}}\,\widehat{\overline{\rho u_j}}}{\hat{\bar{\rho}}}\,, \tag{2.32}$$

which is modeled as:

$$r_{ij}^M = -2\widehat{\overline{\rho}}(C_s\widehat{\overline{\Delta}})^2\sqrt{2\tilde{S}_{ij}\tilde{S}_{ij}\tilde{S}_{ij}^a} + \frac{2\delta_{ij}}{3}\widehat{\overline{\rho}}(C_I\widehat{\overline{\Delta}})^2(2\tilde{S}_{ij}^2)\,, \tag{2.33}$$

where

$$\widehat{\widetilde{S}}_{ij} = \frac{1}{2}(\frac{\partial}{\partial x_j}(\frac{\widehat{\overline{\rho u_i}}}{\widehat{\overline{\rho}}}) + \frac{\partial}{\partial x_i}(\frac{\widehat{\overline{\rho u_j}}}{\widehat{\overline{\rho}}})), \widehat{\widetilde{S}}_{ij}^a = \widehat{\widetilde{S}}_{ij} - \frac{\delta_{ij}}{3}\widehat{\widetilde{S}}_{kk} \tag{2.34}$$

where $\widehat{\widetilde{S}}_{ij} = \frac{1}{2}(\frac{\partial}{\partial x_j}(\frac{\widehat{\overline{\rho u_i}}}{\widehat{\overline{\rho}}}) + \frac{\partial}{\partial x_i}(\frac{\widehat{\overline{\rho u_j}}}{\widehat{\overline{\rho}}})), \widehat{\widetilde{S}}_{ij}^a = \widehat{\widetilde{S}}_{ij} - \frac{\delta_{ij}}{3}\widehat{\widetilde{S}}_{kk}$ Where S_{ij} is the strain rate tensor.

Applying Germao's identity for the Leonard stress tensor, L_{ij}, gives

$$L_{ij} = r_{ij} - \hat{R}_{ij} = (\frac{\widehat{\overline{\rho u_i \rho u_j}}}{\bar{\rho}} - \frac{\widehat{\overline{\rho u_i}}\,\widehat{\overline{\rho u_j}}}{\widehat{\overline{\rho}}} \tag{2.35}$$

The right hand side may be obtained from the filtered variables. Thus, L_{ij} is known. The anisotropic part of the Leonard stress tensor, L_{ij}, can be expressed as:

$$L_{ij}^a = r_{ij}^a - \hat{R}_{ij}^a = 2(C_s\bar{\Delta})^2\bar{\rho}\sqrt{\widehat{2\tilde{S}_i j\tilde{S}_i j\tilde{S}_i j^a}} - 2(C_S\widehat{\overline{\Delta}})^2\widehat{\overline{\rho}}\sqrt{2\widehat{\overline{S}}_i j\widehat{\overline{S}}_i j\widehat{\overline{S}}_i j^a} = C_s^2 M_{ij}^a \tag{2.36}$$

Since $r_{ij}^a - \hat{R}_{ij}^a$ is known and M_{ij} can be calculated, we have 5 independent relations for C_S. This can be solved by least squares. This process can be carried out similarly for C_I, the Prandtl number, and the Schmidt number. [25, 32].

2.3 Rayleigh-Taylor Problem

The RT instability is a fluid instability in which acceleration is directed from a heavy fluid toward a light fluid across a planar interface [3]. Perturbations in the interface position induce pressure perturbations, which cause the interface perturbations to grow. The regions of light fluid penetrating into the heavy are called bubbles; the regions of heavy fluid penetrations into a light ambient are called spikes. As these perturbations grow, secondary Kelvin-Helmholtz instabilities develop on their sides, leading to the formation of mushroom caps at the tips, and vortex roll-up along sides of the bubbles and spikes. By late time, many of these fine scale vortex structures develop and interact, leading to highly complex flow patterns.

An RT instability, at early time, grows exponentially with a growth rate $\gamma = \sqrt{\mathcal{A}g\lambda}$, where g is the acceleration, and λ is the wave length of the perturbation. $\mathcal{A}$ is the Atwood number, defined as $\frac{\rho_h - \rho_l}{\rho_h + \rho_l}$ with ρ_h, ρ_l as the heavy and light fluid densities, respectively. The factor $\mathcal{A}$ acts as a buoyancy correction to the acceleration force g.

Simulating late time RT instabilities is difficult due to the inherent instability of the problem, uncertainty surrounding initial conditions, and due to the numerical smearing of the fluid interface that is present in many Eulerean solvers. Simulating such a system requires a fine mesh, a subgrid turbulence model, and front tracking, as well as correct use of experimental values for fluid transport quantities [19]. LES with front tracking has been shown to be well suited (giving excellent agreement with experimentally observerd growth rates) for Rayleigh-Taylor instabilities [27, 29]. Here we introduce methods for the analysis of stochastic solutions in an LES regime.

2.4 Richtmeyer-Meshkov Instability

The Richtmeyer-Meshkov (RM) instability is a fluid instability induced by passage of a shock wave across a (perturbed) planar interface between two fluids. The instability is similar to RT in the sense that bubbles and spikes grow from the shocked interface, as in RT. In contrast to RT, which produces constant acceleration and velocities increasing in time, early time RM produces constant velocities and amplitudes increasing linearly in time, with slower growth rates at later time.

The initial growth of the interface with the shock impacting at $t = 0$ is given as

$$y(x,t) = a(t)\sin(kx) \tag{2.37}$$

where $a(t)$ is the growth rate as a function of time.

Richtmyer's formula [40] gives the growth rate as:

$$a(t) = k\Delta u \frac{\rho_1 - \rho_2}{\rho_1 + \rho_2} a(0^+) \tag{2.38}$$

where Δu is the difference between the shock and unshocked mean interface velocities. The ρ_i are the post-shocked densities on the two sides of the interface. The incident shock moves from material 2 to material 1. $a(0^+)$ is the perturbation amplitude immediately after the collision of the shock with the material interface. We assume the preshocked amplitude, $a(0^-)$ is small.

2.5 Stochastic Solutions and Second Moments

Due to the inherent instability of turbulent fluid flow, we develop techniques which emphasize the stochastic nature of these kinds of problems. LES grid cell values are cell aver-

ages, not point values. We build on this by proposing a stochastic PDE solution, which is composed of one-point probability density functions on the PDE state variables. Additional physical processes, e.g. combustion chemistry, can be modeled as stochastic processes, dependent on local statistics to determine local reaction rates. Stoichiometric reactions, such as chemical combustion, are known to have reaction rates of the form $\rho_i^a \rho_j^b e^{-A/t}$, where ρ_i and ρ_j are local mean densities of different reactive components. For the case $a = 1 - b$, we can represent the reaction rate using the normalized second moment of the concentration. We study a normalization of the second moment

$$\theta = \frac{\langle F(1-f)\rangle}{\langle f\rangle\langle 1-f\rangle}\,, \tag{2.39}$$

where f is the relative concentration of fluid.

Chapter 3

Wstar: A Comparison Tool for Stochastic Grid Data

In this chapter we explore stochastic solutions to partial differential equations (PDEs). We make use of a theoretical framework which represents grid based data (such as a PDE solution) in terms of probability distributions (over the PDE solution state space). In other words, we reinterpret the PDE solution as a solution of a stochastic PDE. Operationally, this representation is constructed from grid data through a coarsening of the grid and the construction of a PDF in each coarse cell. Theoretically, this description takes the form of a Young measure. We observe that Young measures allow differentiation with respect to space and time (in the sense of distributions) and application of solution functions, so that they can be substituted into a nonlinear PDE as a possible solution. We refer to such a Young measure PDE solution as a stochastic solution. Young measures have found extensive uses in mathematical studies of 1D conservation laws [7, 37, 4, 10] and in descriptions of microphase composite alloys in material science [2].

We provide a process for generating such stochastic solutions from conventional grid based data, and for comparing this data using its stochastic description. A tool is developed to implement this method, and verified by testing for the mesh convergence of a sequence of mesh refined stochastic solutions. We present some of the applications for which this tool has been useful, and pose future applications which may benefit from it.

3.1 The Theory of Stochastic Solutions

3.1.1 Weak* Convergence

We begin by introducing Ω as a problem defined state space. To avoid inessential technical details, we fix Ω to be a closed and bounded subset of $\mathbb{R}^n$. We define $Y = \mathcal{C}(\Omega)$ to be the Banach space of continuous functions on Ω with the *sup* norm. Let $X = Y^*$ be the dual of Y, i.e. the space of linear functionals on Y. This X also defines the space of Radon measures on Ω, i.e. $X = \mathcal{M}(\Omega)$.

For a function $\phi \in Y$ and a measure $d\nu \in X$, we define an expectation operation:

$$\langle \phi \rangle = \int_\Omega \phi(\xi) d\nu(\xi) \tag{3.1}$$

Weak* convergence is the convergence of a sequence of measures in the set X. It is defined as convergence for each linear function $\phi \in Y$. Thus with the sequence of measures $\nu_n \in X$, we say that $\nu_n \to \nu$ in the w* sense, if $\langle \phi, \nu_n \rangle \to \langle \phi, \nu \rangle$ for each $\phi \in Y$.

Each measure is written uniquely as the sum of a positive and negative measure with disjoint support. The norm of each is the absolute value of the measure evaluated on the function identically 1 and the norm of the measure itself is the sum of the norms of its positive and negative parts.

$$d\nu = d\nu^+ + d\nu^- \tag{3.2}$$

$$||d\nu|| = |\int d\nu^+| + |\int d\nu^-| \tag{3.3}$$

As is relevant for existence questions, bounded sets in the Banach space X are weak* compact, meaning that bounded sequences necessarily have convergent subsequences.

We extend this definition to the space of Young measures, $\nu_{\vec{x},t}$, i.e. a function mapping each time-space point to a measure. Thus we denote the sequence as $\nu_{n,\vec{x},t}$ and its limit as $\nu_{\vec{x},t}$. Young measures can be differentiated in the sense of distributions and thus can satisfy a PDE.

Since the use of Young measures is not common in numerical studies, we summarize the key differences between the weak and weak* formulation of a PDE solution in Table 3.1 [21].

3.1.2 Stochastic Comparison of Solutions

We define a stochastic solution of a PDE to be a Young measure solution in which the space-time dependent measures, $\nu_{\vec{x},t}$, are probability measures (i.e. $\langle 1 \rangle = 1$) with the domain Ω. An example of this is a delta function $\delta_{\vec{x},t,\xi_{\vec{x},t}}(\xi)$ concentrated at the value $\xi_{\vec{x},t} \in \Omega$

Table 3.1: Test Functions for Weak and Weak* (Young Measure) Solutions

test function	Weak Solutions	Young Measures
g values multiply g arguments integration domain	$f(\vec{x},t)$ which takes values in Ω $\vec{x},t \in \mathbb{R}^4$ $\mathbb{R}^4$	probabilities $\vec{x},t,\xi \in \mathbb{R}^4 \times \Omega$ $\mathbb{R}^4 \times \Omega$
example	$g(\vec{x},t)$ multiplies $f(\vec{x},t)$ which takes values in Ω	$g(\vec{x},t,\xi)$ multiplies probability

dependent on $\vec{x},t \in \mathbb{R}^4$. This corresponds to a classical weak solution, $f(\vec{x},t) = \xi_{\vec{x},t}$.

As a second example, assume that the Young measures, $\nu_{\vec{x},t}$, are absolutely continuous relative to Lebesgue measure on Ω, and thus defined by PDFs $f(\vec{x},t,\xi)$ with $\nu_{\vec{x},t} = f(\vec{x},t,\xi)d\xi$. We regularize these PDFs by considering their indefinite first integral:

$$\int f(x,t,\xi)d\xi \tag{3.4}$$

namely the CDFs.

The idea of stochastic comparison is to compare these stochastic solutions. For comparison, we take a norm of the difference of Young measures. The comparison result is of the form:

$$E(\vec{x},t) = ||\nu_1(\vec{x},t,\xi) - \nu_1(\vec{x},t,\xi)|| \tag{3.5}$$

where $\xi \in \Omega$ is a random variable in the solution state space, and ν_1 and ν_2 are probability functions from data set 1 and data set 2 respectively. The norm in (3.5) has yet to be specified, but many choices are possible. Conventional function space norms on $\mathcal{M}(\Omega)$ are then applied to the CDFs in Eqn. (3.5). We typically compose a norm over X (state space measures) with a norm over functions on $\mathbb{R}^4$ representing the ν_i in terms of their CDFs.

3.1.3 Large Eddy Simulations

We consider stochastic solutions to LES (see §2.2) turbulent mixing problems. Engrained in LES is the notion that subgrid fluctuations resemble grid level fluctuations. The conceptual framework seems to be especially appropriate here, as LES are (by definition) under resolved, and have fluctuations below as well as at and above the grid level. We interpret this to mean that the statistics collected at the grid level should be representative of the statistical distributions hiding beneath the grid level. Of course, this is the basic idea of turbulent self similarity and Kolmogorov's scaling laws.

We propose this technique for the comparisons of data: from distinct meshes (e.g. in

a mesh convergence study), from distinct physics parameters (in a parameter dependence study), or from numerical to experimental data (in a validation study).

The idea of LES is that fluctuations at or above the grid level (observed numerically) can be used to predict fluctuations below grid level (not observed numerically). We extend this to the stochastic assumption, that although the solution is fluctuating rapidly, the statistical laws that govern this fluctuation are changing only slowly. Thus we seek to capture the slow variation of the statistical laws (on a coarse grid) and the statistical laws that define the rapid fluctuations (as a probability measure).

3.2 Wstar

The mesh level comparisons performed can be applied to many types of comparisons in which simple differencing or averaging fail to capture important qualities inherent in stochastic phenomena. We discuss methods for stochastic convergence of mesh based simulations and we illustrate this concept with particular applications to LES turbulent mixing simulations.

The first problem which wstar addresses is the construction of a stochastic solution from a (non-stochastic) point-valued dataset. Such a solution is constructed by relaxing the fine grid dependence on space and time. To this end, we form a coarse grid (supergrid), composed of unions of mesh cells (supercells) and resolve the space-time aspect of the data on the coarse grid only. Within a single supercell, the collection of solution values (from the many fine mesh cells in a single supercell) is treated as a (space-time independent) random sample of state values (e.g. density, pressure, momentum, etc.).

The strength of a comparison based on this reorganization of the solution data is that there is no averaging. Thus variations and fluctuations are preserved. Although coarsening is necessary, there is not a loss of information. Rather, data is reinterpreted. Nonlinear functionals of the data (e.g. combustion rates in a chemistry problem), which respond to the fluctuations correctly, converge along with the original sequence [22].

3.2.1 The Algorithm

The wstar tool is designed to perform a comparison between two different data sets (varying parameters, refinement, etc.) following the theory presented in §3.1. That is, it computes a (discrete approximation to) a Young measure for each data set and compares their difference as in Eqn. 3.5. We tend to prefer L_1 for the norm. A comparison of L_1 with L_∞ is shown for our verification dataset in Table 3.2.

The inputs to a wstar comparison are two raw data sets and a coarsening factor for each to define the supercells for each set. Also required is a bin size, or state space discretization. Wstar is implemented in two steps. The first step is to process the raw data into a stochastic (Young measure) solution. This is done separately for each data set that is to be compared. In the second step, the Young measures are compared by a norm difference of measures in each supercell (by Eqn. 3.5), and the numerical result is recorded at the supercell location. This creates a supercell grid of comparison (norm of difference) results. Note that the result is a Young measure, but differences of probability measures are not typically probability measures. This result can be visualized and can be integrated to define an overall L_1 norm of the difference.

To accomplish this task, during the second step, an injection is defined to map the supercell defined data from one comparison grid onto the other. There is no requirement that the CDFs have the same state space discretization (bin size) or same space-time discretization (supercell size). We refine the coarser mesh by a piecewise constant injection into the finer mesh. The reverse injection is also possible, by construction of the union of the random samples and injecting this union of data into the coarser stochastic solution.

3.2.1.1 Computing the CDF

We begin by constructing the PDF in a coarse grid cell. To construct the PDF, we introduce for each supercell, a mesh discretization of Ω ("frequency bins" in state space). Within a supercell, the fine grained data is treated as a random sample, i.e. with no notion of spacial or temporal dependence. In each supercell, we count the number of sample points from the fine data that lie in each bin. The integer counts are normalized to yield a total mass of 1. This numerical piecewise-constant PDF is integrated to obtain a piecewise linear CDF (as in Eqn. 3.4) which is used for further analysis.

The degree of discretization, i.e. the bin size, is a state space convergence issue quite different from conventional space-time grid resolution issues. [22]. There is a trade off between the numerical integration error (in CDF and in the differencing) due to a coarse bin size, as no knowledge of solution values internal to a bin is retained; and the statistical sampling error due to the finite sample size for each bin. Decreasing the bin size improves integration accuracy, but leaves less sample data in each coarse cell, increasing statistical error. Given a bin size, δb with volume δb^3 and a supercell sample size N, the amount of data per bin is about $\frac{N}{\delta b^3}$. Assuming smooth PDFs and CDFs, the integration error is $O(\delta b)$. The statistical sampling error is $O((N/\delta b)^{1/2})$. These two sources of error must be balanced using standard statistical tests to evaluate this trade off.

3.2.1.2 Comparison

At this stage, a discrete Young measure has been defined over the supergrid represented as a CDF in each supercell with $\vec{x}, t$ defined by supercell location. We obtain a value in each supercell that is the difference of the two CDFs taken in the L_1 norm. We can integrate this field over the spatial domain to to yield an overall mesh norm i.e.

$$L = \int_D E(\vec{x}) d\vec{x} \tag{3.6}$$

Where $E(\vec{x})$ is defined in Eqn. 3.5. [22]

An example of spatial arrays of PDFs and CDFs to define a Young measure description of simulation data is given in Fig. 3.2 in §3.3.1.1.

3.2.2 Implementation and Usage

Wstar is designed as a post processing tool. It is written in Python version 2.6. This construction gives it cross platform support as it rides on the portability of the Python language, which is a standard in computing. Wstar supports IO for files in VTK structured grid format. It is designed, moreover, to simplify adding IO support for different file types. See the Wstar Manual in the appendix for details on support for various file types.

The post processing is executed in two scripts:

The first script, `wstar.py`, parses an input file and set of data files to generate the PDFs on the coarse grained grid of supercells. The coarse graining factors and bin sizes are inputs at this stage. The PDF is stored as an array of bin counts as data is read in from the data files. As each datum is read, it is determined to which supercell it belongs, and then to which bin it belongs. The corresponding data counter is incremented for that cell and bin, and the process continues until all of the data is read in this way. At this stage the PDF is normalized, and the frequency counts are replaced with probabilities in a corresponding array. This data is integrated into a piecewise linear CDF (as in Eqn. 3.4), and this data is then stored in an intermediate file. This is done individually for each data set to be compared [22].

The second script, `wstar_compare.py`, generates the final comparison. It takes as input two CDF supergrids made via the previous instructions, and generates the comparison as its output. The comparison contains L_1 norms of CDF differences, as well as other statistical information such as means and variances of state variables from each. VTK structured grid is used to generate output plots of these [22]. Alternative formats for IO should be easy to implement following the manual.

3.3 Applications

The foremost feature of wstar is the ability to analyze and compare stochastic (Young measure) data sets, such as solutions of PDEs. Another noteworthy feature in these examples is the ability to tease out valid scientific conclusions from data in which simple inspection is not sufficient, and the conclusions are somewhat buried in the statistical noise.

The Young measure aspect of the data is emphasized in our first set of figures (see Fig. 3.1 and Fig. 3.2). In a following analysis we illustrate the ability of wstar to analyze higher statistical moments. Here we study second moments.

3.3.1 Rayleigh-Taylor Turbulent Mixing

The RT instability is an acceleration driven fluid instability described in §2.3. Here we investigate an RT simulation using LES, dynamic SGS and front tracking. We will show mesh convergence of the CDFs using wstar. Initial conditions for Rayleigh-Taylor mixing experiments have been somewhat mysterious.

3.3.1.1 Stochastic Mesh Convergence

We demonstrate verification for wstar through a mesh convergence study, applying weak* convergence (using wstar) to a Rayleigh-Taylor instability problem, following previously published material [21]. The raw data to be compared is shown in Fig. 3.1 in the left frame, concentration in a slice through the midplane for three grid levels, coarse, medium, and fine. While these plots suggest some level of convergence, the true measure of convergence emerges only from our wstar analysis. In the right frame, are shown the differences of the CDFs. The medium to fine comparison shows systematically smaller CDF differences than the coarse to fine comparison. Thus we demonstrate the ability of w* to make obvious and clearcut facts that are only weakly discernible from direct examination of the data. We also use this data set to explore the setting of various parameters in the wstar algorithm.

We study mesh convergence using a 8×2 array of supercells to define PDFs and CDFs. At the latest time completed for the fine grid, we compare the PDFs and CDFs on the coarse to fine and medium to fine grids. We compute the L_1 norm of the pairwise differences for each of the 8×2 PDFs or CDFs. These differences yield an 8×2 array of norms, i.e. numbers, which is plotted in Fig. 3.1, in the right frame.

In Table 3.2, we apply the L_1 norm to Eqn. (3.5) using PDFs and CDFs on a simplified solution state space (Ω) which is the concentration of light fluid. To simplify further, we apply L_1 and L_∞ norms in space (x, y) to yield a single number for the entire data set. We

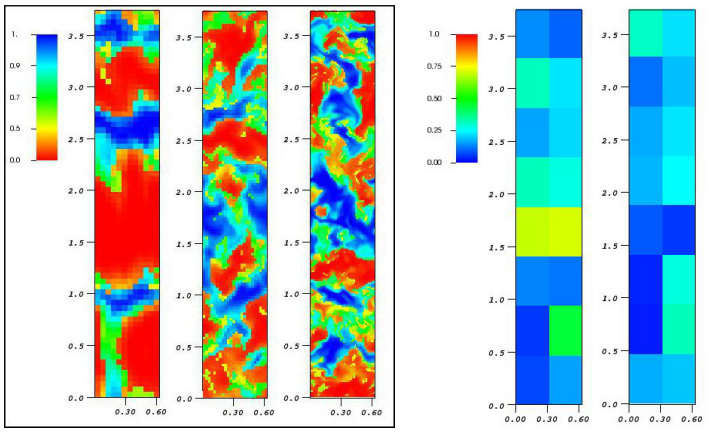

Figure 3.1: RT mixing on 3 grids with stochastic comparison. Left: Plot of heavy fluid concentration at the midplane, $t = 50$. Coarse grid (left), medium grid (center), fine grid (right) Right: Spatial array of L_1 norms of CDF mesh differences for heavy fluid concentrations at the midplane. Coarse to fine (left). Medium to fine (right).

explore the effects of [21]

- the mesh,
- PDF vs. CDF,
- L_1 vs. L_∞ for a spatial norm and
- the size of the supercells.

To construct the PDF, we divide the concentration, $\xi \in [0, 1]$ into 5 bins. The result of this exercise is an 8×2 array of PDFs, each represented in the form of a bar graph. The array is a graphical presentation of the Young measure at the fixed z, t value. See Fig. 3.2. From this array of PDFs, we can observe some level of coherence or continuity in the spatial arrangement of the PDFs, in that the central supercells have a strong heavy fluid concentration, while near the top and bottom, there is more of a mixed cell concentration.

The convergence suggested in Fig. 3.1, is now a clear trend in Table 3.2. Generally L_1 norms show better convergence, and generally there is a minimum size for the supercell to obtain useful convergence. Since our convergence properties are documented for the medium grid (through comparison to the fine grid), we can speculate that the errors at the fine grid level would be smaller.

Table 3.2: Summary norm comparison of convergence for heavy fluid concentration PDF and CDF at fixed values of z, t In each supercell, an L_1 norm is applied to the difference of the PDFs or CDFs; this x, y dependent set of norms is measured by an L_1 or L_∞ norm. The larger supercell sizes, the last four columns of the table, cover the entire y domain. In this case, the space-time localization of the PDFs/CDFs are in x, z, t only. We observe convergence for CDFs; while the PDF error is decreasing, further refinement will be needed for usefully converged PDF errors. We see that a coarsening of the supercell resolution (increase of the supercell size) to 18×12 coarse grid cells per supercell is needed to obtain single digit convergence errors.

coarse grid supercell size	$9 \times 6 \times 1$		$18 \times 12 \times 1$		$36 \times 12 \times 1$	
mesh comparison	L_1 norm	L_∞ norm	L_1 norm	L_∞ norm	L_1 norm	L_∞ norm
CDFs: coarse to fine	0.26	0.98	0.16	0.48	0.15	0.39
CDFs: medium to fine	0.18	0.54	0.08	0.16	0.03	0.10
PDFs: coarse to fine	0.93	4.89	0.59	2.40	0.54	1.98
PDFs: medium to fine	0.64	2.66	0.30	0.82	0.15	0.52

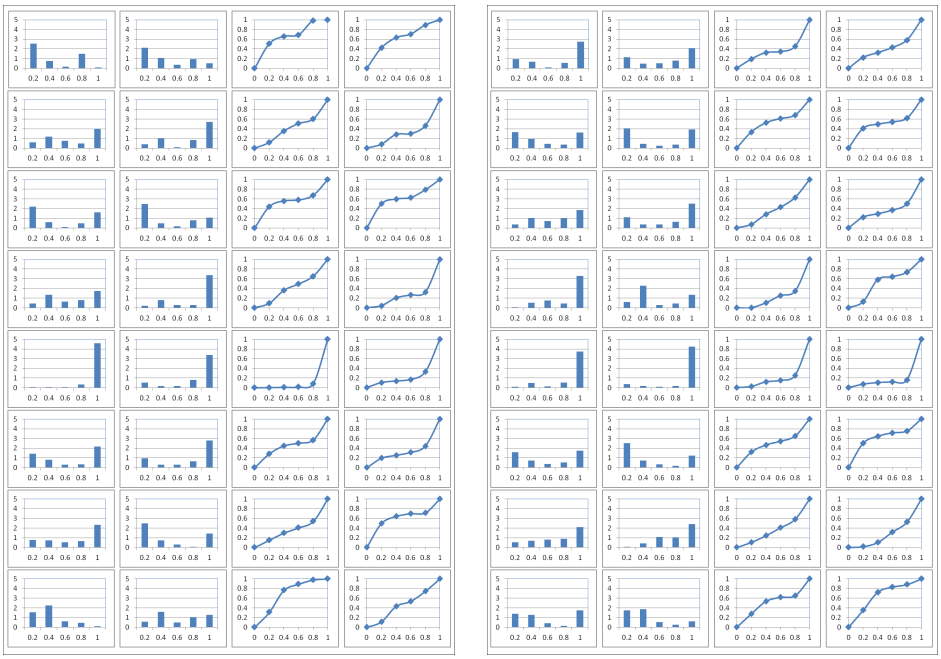

Figure 3.2: Spatial array of heavy fluid concentrations at t = 50, for z in the midplane, as PDFs (bar graphs) and as CDFs (line graphs), Left: medium grid. Right: fine grid.

3.3.1.2 Second Moments

We verify wstar by showing convergence of second moments. We follow the previously published text [21]. The measure by ν of the functional $f : \Omega \to \mathbb{R}$, denoted $\langle f \rangle$, is defined

in Eqn. 3.1. We define the normalized second moment of f, to be:

$$\frac{\langle f(1-f)\rangle}{\langle f\rangle\langle 1-f\rangle} \tag{3.7}$$

As these are more slowly convergent, we use a single supercell, and do not emphasize the Young measure (coarse grid spatial localization) aspect of the data. Rather we emphasize the ability of wstar to handle higher statistical moments. The algorithm can accomplish both goals at once, but the data we analyze is not strong enough to allow such an analysis.

We show the convergence under mesh refinement of the second moments for species concentration and velocities, quantities of interest in a miscible Rayleigh-Taylor experiment [43]. We apply wstar with one supercell (treating the entire dataset as a single sample). The same tools are applied to velocities (giving the Reynolds stress), but as the conclusions for the use of w* are similar to those for concentrations, we refer to [21] for those results, omitting them here. Since the quantities we report were not measured experimentally, this study is verification only, not validation. A related simulation study [29] includes comparison to the water channel experiments [34, 35], in which the second moments were measured, and thus for which validation was studied.

It is commonly believed (and observed in numerical studies) that fluctuating quantities obey a type of Kolmogorov scaling law. This property, if correct, implies that the fluctuations are represented by a convergent integral, and should be convergent under mesh refinement. Thus the convergence we report here should not be a surprise. Still, the documentation provides some new information: the level of refinement needed to observe convergent behavior. We generally observe satisfactory convergence through comparison between the medium and the finest of the three grids considered here, and unsatisfactory (poor agreement with the refined grid) properties for the coarsest grid. The limits at late time encounter a varying loss of statistical resolution due to the diminished number of statistically independent degrees of freedom at late time. The three grids have a size 520 to 130 microns (4 to 8 to 16 cells per elementary initial wave length). Of these, we have generally used the medium grid in our previous simulations, while the coarse grid is commonly favored in RT studies [6]. All second moments reported here represent mid plane values, i.e. a slice $z = \text{const}$ from the center of the mixing zone with t fixed, and are averaged over all x, y values. All second moment plots were constructed from wstar, using the full domain as a single supercell.

The second moments of concentration, normalized, define the molecular mixing correlation $\theta = \langle f(1-f)\rangle / \langle f\rangle\langle 1-f\rangle$. Our value for $\theta \approx 0.8$ is consistent with values obtained numerically in related problems by others. However, significantly smaller θ values were observed in the similar fresh-salt water miscible experiments [34, 35]. Since these differences

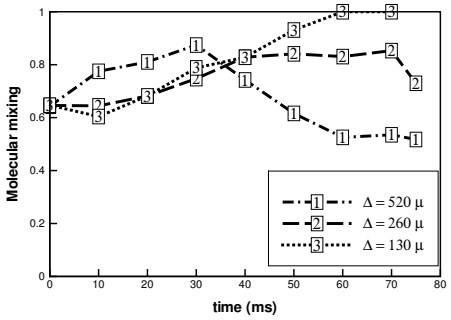

Figure 3.3: Plot of θ vs. time for a numerical simulation of the experiment [43] #112. Three grid levels shown: $\Delta x = 520, 260, 130\mu$. The data has been collected at the mid plane value of z, with an average over all of x, y.

are observed even at very early times, we can attribute the differences to initial conditions, specifically to the thickness of the initial diffusion layer. Fig. 3.3 displays numerical results for convergence of θ which model experiment [43], #112, with three levels of mesh refinement.

3.3.2 Richtmeyer-Meshkov Turbulent Mixing

We apply the wstar tool to show mesh convergence in an RM instability study (see §2.4 about RM instability). Again, the strength of wstar is its ability to extract from the data clear signals for properties otherwise lost in the noise of fluctuations. We demonstrate here one such example and refer to other examples in [33]. In summary, we present evidence for mesh convergence of the CDFs below, and refer to [33] for details of other studies which show a weak dependence on the Reynolds number in the limit $Re \to \infty$. For this conclusion, we need to compare two small quantities extracted from the statistical noise, namely the numerical convergence rates and the signal from Re variation. The point of this analysis is that the numerical convergence errors were (somewhat) smaller than the physics based Re variation effects, indicating that the later was in fact a real effect and not a mesh error artifact. These same references also study the variation in the solution caused by variation of the numerical algorithm (in the form of modified subgrid model coefficients). Here the contrast was larger and clearly visible even without a wstar analysis. This conclusion in this case is the solution non-uniqueness, and its dependence on the numerical algorithm. It should be noted that this conclusion is itself striking, as it flies in the face of conventional notions of numerical predictive science.

In Fig. 3.4, we plot the L_1 norm of convergence errors in the concentration CDFs at $Re = 6 \times 10^7$. The picture represents a time after reshock. The turbulence is well developed

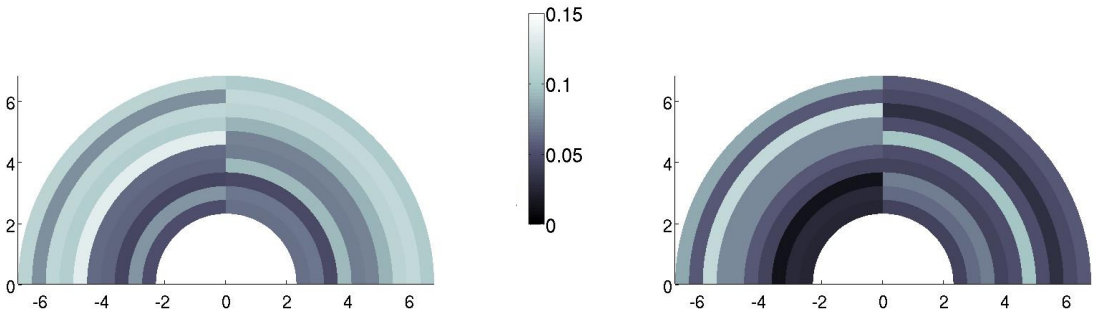

Figure 3.4: Mesh convergence of the concentration CDFs is demonstrated visually, with data from [33].
The left frame compares the coarse to fine mesh; the right frame compares the medium to fine mesh. Colors indicate the level of CDF mesh error in each supercell. Expressing this convergence as an L_1 norm, the order of convergence is 1/2.

at this time and the high values of Re are reflective of the Kolmogorov hypothesis. In the left frame, the coarse grid is compared with the fine grid and in the right frame, the medium grid is compared with the fine grid. This figure is a visualization of the w* convergence assessment. The numerical values associated with the plots of Fig 1.5 can be integrated to obtain an overall norm of convergence, and this is compared, coarse to fine and medium grid to fine. The order of the mesh convergence, using an L_1 norm, is about 1/2, which is slightly better than order of convergence of joint concentration-temperature CDFs [33].

3.4 Conclusion

The wstar tool has been verified to demonstrate mesh convergence in a turbulent mixing regime via second moments. We hope to see it used in the future to demonstrate LES convergence quantitatively. We also believe that validation studies in the turbulent regime should use this tool. Moreover, the notion of a stochastic solution as a representation of a turbulent mixing solution has a variety of other potential uses that still stand to be brought to light. These techniques can be embedded in a simulation tool for predicting nonlinear functions on state data, i.e. combustion or other chemistry reactions. Wstar still stands to benefit from some enhancements. Parallel post processing would greatly enhance performance. Alternative geometry would allow a broader scope of use.

One issue is that there is relatively little experimental data to compare fluctuations, and only limited ability to validate the properties of PDEs presented here. We hope that experimentalists will take up this form of data analysis, in addition to computational scientists. Experimental scientists report repeatable data, and at first glance, most turbulent

data seems not to be repeatable due to its rapid fluctuations aspects. However, with the stochastic reinterpretation of data, the experimental measurements should be repeatable in the fluctuating regime, and if reported, would provide a basis for validation, thus allowing a significant advance in scientific analysis of turbulence.

Chapter 4

FTI (Front Tracking Interface)

Front tracking is a numerical method which introduces dynamically evolving fronts into an Eulerean grid based simulation. The use of front tracking is to preserve and maintain sharp discontinuities or steep gradients, which are often poorly resolved by Eulerian methods. Block stencil based operations near the front where the block stencil contains states with differing materials or properties produces a smearing of discontinuities or gradients. The intention of front tracking is to minimize this numerical diffusion. Front tracking in an Eulerean simulation framework can be considered to be the ultimate Arbitrary Lagrangian Eulerean (ALE) algorithm in that it is minimally Lagrangian. The Lagrangian aspects of front tracking are robust and better able to survive the late time mesh distortion from complex flows that normally limits ALE effectiveness. The common ALE recourse to Eulerian dynamics is significantly postponed or totally eliminated with the use of front tracking.

A standardized application programming interface (API) for porting front tracking models into arbitrary simulation codes is introduced. We discuss models for advecting the front according to laws of physics as well as methods for communicating front information into the solver. Front Tracking Interface (FTI) is an API designed to make front tracking conveniently accessible to new or existing simulation tools. FTI has only a small number of essential routines, a fact that contributes to the simplicity of its use. The FTI routines are divided into

1. Server Routines, and

2. Client Routines.

The server routines are provided in the FTI server (FronTier, §4.4.1). They allow manipulation and queries of the front geometry. Routines which are delegated to the client require knowledge of the physics, simulation grid, or states in the simulation. While the goal of

this design is to provide a generalized package for tracking a dynamic geometric front, the discussion here is focused on computational physics (e.g., fluid dynamics).

This work follows the FronTier Lite project, which also provides an API for front tracking. FronTier Lite provides robust methods for interfacing with the front geometry. FronTier Lite is quite versatile and robust due to its complete independence from underlying physics. The aim of FTI is provide a simpler interface to front tracking that is targeted to physics applications, and to provide clear and concise methods for interfacing various types of physical models to the front.

We target a broad range of possible client applications. Thus, FTI is based on a client-server model in which the server manages the front, but requires the client to implement routines dependent on client data structures and physics. Implementations of the FTI client are provided in cFluid (§4.4.2.1) and in the UChicago high energy computational fluid dynamics (CFD) code, FLASH (§4.4.2.2). We also assist in the development of implementations of the FTI client in the weather simulation code WRF, and an electrocardiac simulation code. (See Table 4.4 for a list of clients and their current statuses.)

4.1 Front Tracking Conceptual Framework

The essence of front tracking is handling the dynamic motion and topological changes of a front. This physics-free conceptual framework allows a variety of uses (e.g. weather modeling in §4.4.2.3, elastic-plastic slide surfaces in §4.5.4.3). Reference implementations for the Navier-Stokes equations with a WENO numerical solver are provided. The front, represented as an unstructured triangulated mesh, is assumed to be a closed surface, or an open surface with boundary constrained to lie on the domain boundary. It is stored using data structures from computational geometry: discrete points with some connectivity information, i.e., edges and faces. The front, as constructed above, separates the space into distinct regions. We define the term 'component' as an indicator for these regions. Regions on different sides of the front are said to have different components.

The two main tasks in front tracking are:

1. Propagation of the front

2. Propagation of the interior (non-front) states near the front

It is assumed that the interior state update involves a stencil of adjacent states, with the solver operating on that stencil. We introduce a modification to this algorithm for a stencil which crosses the front, as this crossing is exactly the condition that causes numerical mixing.

FTI provides a detection algorithm to find stencils which cross the front and a ghost state extension to modify the stencil states beyond the front crossing. The client solver acts in the usual manner on this modified stencil. We also have a vectorized version of this algorithm that works well in a vectorized dimensionally split solver.

We require an algorithm to compute the solution state at arbitrary locations in the domain. These interpolated states come from the interior grid but use only states of a specified component. This component aware interpolation is used for computing unmixed interpolated states near the front.

We offer two modes of operation:

1. state-free tracking,

2. and dynamic state tracking.

In state-free tracking, states are computed at the front using component aware interpolation (§4.1.3) and solving the Riemann problem (§4.1.4) at the front. We define the front states (§4.1.5) to be the mid states of this Riemann problem solved at the front point using two interpolated states as the two outer states (one from each component). The coupling between the two separate regions separated by the front occurs through the Riemann problem connecting them.

In dynamic-state tracking, solution states are also stored at the discrete points on the front. They can, of course, be interpolated to give a solution state anywhere on the front. They are updated during the front propagation step.

4.1.1 Front Geometry

The front is represented in 3D space as a set of triangles, each of which contain points. Triangles that share a common edge are said to be adjacent, and that adjacency information is also stored. This set is partitioned into surfaces, each of which must either have edges that meet the domain boundary (open surface) or be entirely closed. Thus, as presently implemented, each surface creates a bipartite partition of the spatial domain.

Efforts in front tracking began with the grid based (GB) front. The points of this front are constrained to lie on the grid lines of the simulation grid. This constraint gives a stable, evenly distributed discretization of the surface. However, it is clear that this method must be improved to gain subgrid resolution of the front. The next generation, now in use, uses the locally grid based (LGB) front. This front is allowed to move freely, unconstrained by (in fact, entirely agnostic to) the simulation grid, with one exception. When the front triangles intersect each other, topological changes must be computed and the interface must

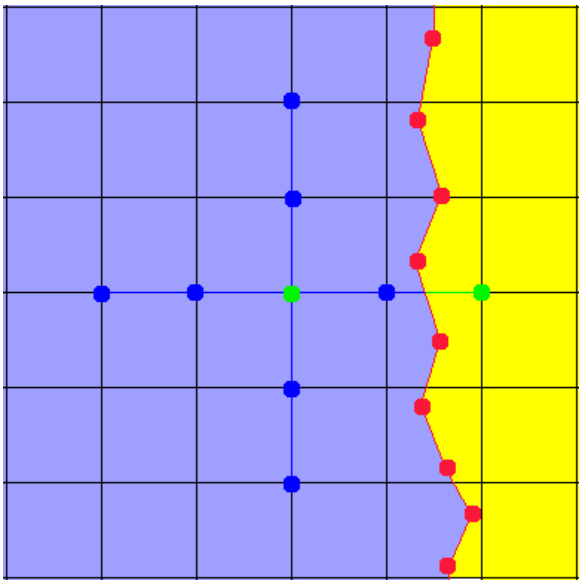

Figure 4.1: Illustration of a component grid with a front crossing solution stencil. Cells in the pure yellow and pure blue regions are assigned different components. This grid is dual to the simulation grid, so component and state information is defined at cell corners.

be reconstructed. This is done by creating a GB representation of the front local to the points of intersection, then performing a GB reconstruction to produce a topologically valid front, then rejoining the GB region to the rest of the front. This algorithm gives a high resolution, efficient front, with the robustness of being able to handle changing topology.

4.1.2 Components

Components are computed and stored at grid cell centers (shown in Fig. 4.1). This requires determining components for a cell center from its coordinates. This calculation is based on finding the nearest front point. This expensive call is used to initiate the marching front method (see below).

These components are used in the component aware interpolation model (§4.2.3.3). It is also used in the solver to determine the material region in which a given mesh cell lies. Other uses for component in a simulation code may be to assign physical models (e.g. EOS) based on component, as different materials may have different models associated with them.

4.1.2.1 Marching Front Algorithm

This algorithm begins with a grid in which all components have been set to some placeholder value (e.g. 0). First we iterate over the triangles of the front and mark, with another

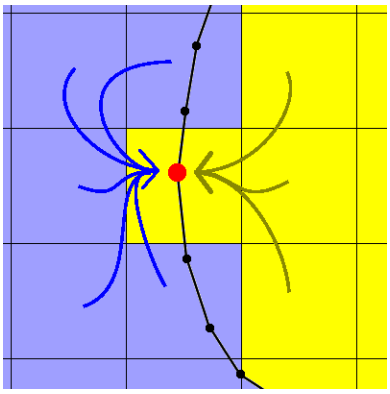

Figure 4.2: A conceptual view of component aware interpolation

placeholder value (called ONFRONT) which marks the cells which are cut by the front.

Now we can start by making an expensive component query on a particular grid cell (whose component is unknown). The front is representative of a surface, and so two adjacent cells cannot have different component unless there is an ONFRONT cell between them. By this assumption, adjacent cells can be filled in with the same component (and adjacent to those, and adjacent to those, etc.) until an ONFRONT cell is reached. This will fill in the component of an entire region with only one expensive component lookup. We can then continue onto another cell of unknown component until all of the components are known.

4.1.3 Interpolation

The client is required to compute the solution state at an arbitrary point in the simulation domain. Many simulation tools already have grid interpolation routines. However, we require an interpolation in regions near the interface between different components with distinct material properties. Therefore, we require an interpolation that takes this into account, computing the solution state at a point based only on adjacent states of a specified component. The amount of available information to the interpolator in this type of use is variable. It is desirable to use high resolution interpolation (usually bilinear or trilinear here); however it is possible that there is not enough available information. It is necessary to reduce the interpolation (down to constant order if there is only one datum available) to obtain the desired result.

4.1.4 The Riemann Problem

The Riemann problem specifies Cauchy data depending on a single spatial coordinate, and in this coordinate the data is piecewise constant with a single discontinuity at the origin. The

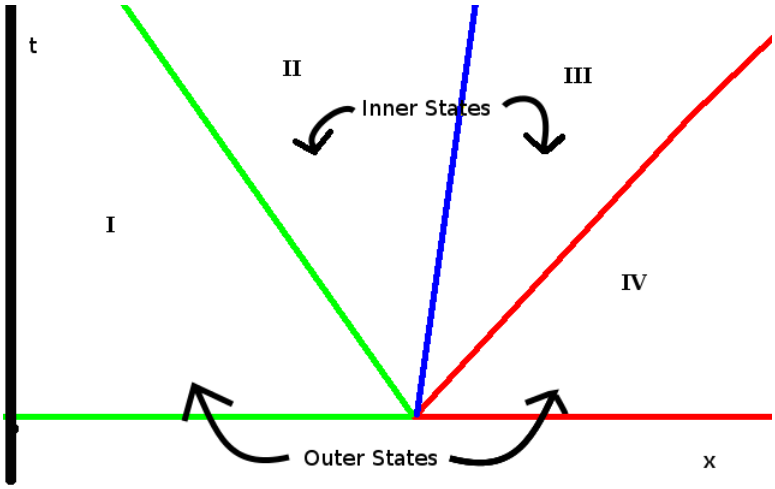

Figure 4.3: An illustration of the Riemann solution. Red and green are simple waves. Blue is the contact discontinuity. The 4 constant state regions are labeled in Roman numerals.

Riemann solution for the Euler equations of a compressible fluid has four constant regions (see Fig. 4.3) separated by three waves. The two outer regions are outside of the influence of the fluid interaction, whose effects travel at finite speed. The two inner regions are in contact, and are in pressure equilibrium. The waves separating the outer regions from the inner are shock or rarefaction waves. The central wave is a contact surface between the two fluids [5].

The initial conditions for the Riemann problem are two constant regions (I and IV) separated by a contact discontinuity. Since information travels out from the contact at finite speed, the initial constant regions continue to exist for all time. However, as soon as the system becomes dynamic, two inner regions will form around the contact (II and III). The mid states for the inner regions are in pressure equilibrium. The boundary between the inner state and outer state represents a simple wave, either rarefaction or shock, propagating at finite speed. The waves must lie on the characteristics. The Euler equation characteristics are:

$$C^+ : \frac{dx}{dt} = u + c\,, \tag{4.1}$$

$$C^- : \frac{dx}{dt} = u - c\,, \text{and} \tag{4.2}$$

$$C^0 : \frac{dx}{dt} = u \tag{4.3}$$

where c is the sound speed. The contact wave lies on the C^0 characteristic. These are straight lines in the Riemann problem, which explains the straight line region boundaries in Fig. 4.3 [5].

To relate this back to front tracking, the front represents a contact discontinuity, and the discontinuous solution near the front is modeled as the mid states in Riemann problems solved at the front. This choice is physics and problem dependent, and other choices (e.g. a flame front or a dynamic phase transition boundary) are possible.

4.1.5 Front States

On the points of the front mesh (which are generally not at cell centers), the front states are defined. These are fluid states which specify the solution at the front. Each front point has two front states, one from each material meeting at the interface. We provide two means of assigning front states. Both methods are rooted in the assumption that the local solution near the front is the solution to a local Riemann problem.

4.1.5.1 State Free Tracking

In state-free tracking mode, the front states are computed from interior states as needed during the simulation, using the following algorithm. Component aware interpolation (§4.2.3.3) is called at the front point for each of the two components that the front point separates. This gives gives two states, one from each component, at the front point, S_1 and S_2, the state resulting from components 1 and 2 respectively. The Riemann problem (§4.1.4) is solved normal to the front using the two interpolated states as outer states. The mid states of the Riemann solution, $\hat{S}_1$ and $\hat{S}_2$ gives the two front states at the point. This gives a simpler front tracking client than the dynamic state model. Evaluation of the relative accuracy of these algorithms is in progress.

4.1.5.2 Dynamic State Tracking

In dynamic-state tracking mode, the front state is a dynamic variable stored at each front point. These are initialized using the front state algorithm from 4.1.5.1, and updated during the front advection step. As physics dependent variables, the front states are updated using operator split dynamics in the normal and tangential directions during front propagation. The guiding physics and basic algorithm are shown in §4.1.7.2.1 and §4.1.7.2.2 for normal (MOC) and tangential update respectively.

4.1.6 Initialization

Initialization is performed by the server by creating a meshed surface and setting the initial front states on that surface. The surface is initialized from a client supplied level

set function, ϕ. The front is created as the isosurface at $\phi(\vec{x}) = 0$ via the marching cubes (or marching squares in 2D) algorithm. This creates a grid based interface, using a regular Cartesian grid. The dimensions of the grid are chosen such that the initial front has sufficient granularity for its intended use, generally taking the grid spacing from the interior state mesh. This grid spacing is used only for initialization, and for the grid based untangle algorithm (§4.3.2).

4.1.7 Front Propagation

To accomplish the front advection, velocities at and near the front points are needed. The front state velocity (defined below in each operation mode) gives a first order definition of the front motion. Both dynamic state and state free modes make use of the front state for their front point updates.

4.1.7.1 State-Free Mode

The normal velocity is continuous in the Riemann problem; therefore this velocity is well defined at the front point. The point is advected by a second order Runge-Kutta scheme using the front state velocity. In state-free front tracking, we use the front state algorithm defined in §4.1.5 to get the velocity for both the predictor and corrector steps. RK2 is applied to the following:

$$v(\vec{x}, t) = \text{velocity}(\text{frontstate}(\vec{x}, t)). \tag{4.4}$$

and solving

$$\frac{d\vec{x}}{dt} = V(\vec{x}, t) \tag{4.5}$$

as shown in [9].

4.1.7.2 Dynamic-State Mode

In dynamic-state front tracking, the front states are stored and must be updated, as well as the front positions. The update is performed first in the normal direction during point propagation. This algorithm uses a method of characteristics solver, and a second order predictor corrector method (RK2). After the point is advected, there is a tangential update.

4.1.7.2.1 Normal (MOC) Update The normal update is done at the same time as the point advection. When the predictor is applied, the front point takes a new position. The Riemann characteristics (Eqns. 4.1 and 4.2) are found and traced back to the previous time

level. An ODE for the Riemann invariants [5] is solved using the interpolated state from the previous time level at the characteristic. This gives a new front state for the corrector method. After the corrector is applied, the Riemann problem is solved using the new front states as input, to ensure consistency.

4.1.7.2.2 Tangential Update The tangential front state update is performed after the front points are advected to the new time level, and the front states have been updated by the normal update. The tangential effects are computed using the interior PDE solver (the same one that is used to update the interior states). To perform the update at each point, the front is projected onto the tangent plane around the point and the nearby front states give a 2D unstructured solution. This unstructured solution is interpolated onto a stencil compatible with the interior solver, which calculates the flux to perform the update.

4.1.8 Interior State Propagation

A PDE solver is used to update the interior states, as usual, using a finite stencil and a numerical scheme to update each interior point. However, solver stencils may be cut by the front, i.e., the stencil may contain data from two different components (with differing material properties). To maintain consistency (sharpness), the solver is modified to detect cut stencils, and a ghost state algorithm is used.

The ghost fluid method is the model by which front tracking maintains sharpness in the data field of the simulation [13]. Numerical diffusion usually occurs when updating points near the material interface, as data from one material is used in the stencil for an update at a point located in another material. Using FTI, one can identify where a stencil crosses the material interface, and provide a model for substituting single component data into the stencil for the solve.

Conceptually, the interior solver sees the state from one side of the front only. The purpose of the ghost fluid method is to accomplish this goal. The physics of coupling across the front is handled by the front state Riemann problem only (defined in §4.1.4).

4.1.8.1 Dynamic Component Change

An interior grid cell may find that the front has crossed over it during a time step. At the new time level, it lies on the other side of the front. This is done by using component aware interpolation on the cell center coordinates using the new component. The old state is discarded. We find that the conservation of mass violation is a small effect, and will be addressed in future work (see §4.5.1).

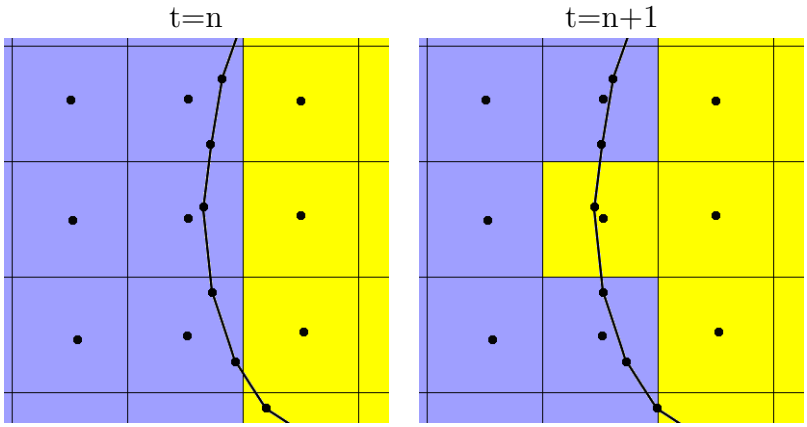

Figure 4.4: An example of a grid cell with dynamically changing component

4.2 FTI Client Implementation

Implementation of the front tracking method can be summarized by the following four modifications to a solver based physics application:

- Initialize
  - Call during initialization
- Propagate Front Points
  - Call in time step loop
- Dynamic Component Update for New Time Level
  - Call in time step loop after propagate front points
- Modify Stencil for Interior State Update
  - call when creating solver stencil

Since FTI cannot know or enforce conditions on the client data model, it is the task of the client to generate, advect (in dynamic-state mode), and interpolate on front states. The front states are the crux of the coupling between the interior dynamics and the front dynamics.

4.2.1 Component Update

The client performs the marching front algorithm, as described in §4.1.2.1. The server provides routines for detecting cut cells, for determining the component at the start of the

algorithm. The components are stored by the client on the grid cell centers. The dynamic component change (§4.1.8.1) is also calculated during this step.

4.2.2 Modify Stencil

To find the state values for the stencil point immediately following the interface, we first find where the stencil crosses the front. The front state having the same component as the stencil (described in §4.2) is computed for that point. It is this state that is used in the solution stencil (by constant extrapolation) for all values needed across the front [13]. Conceptually, in this mode, the interior solver operates purely on a single side of the front (one material only) and never crosses the front. All cross-front information comes via the Riemann solution in the front state definition.

The algorithm in this direction split, vectorized solver is as follows. We begin constructing sweep vectors. As we pack data into the vector, we check the component of the new data. If the component is the same as the data already loaded, we append it to the vector and continue. If the component changes, we stop. This gives a sweep vector of homogeneous material. We only need to add buffer states onto the end of these sweep vectors. If the edge of the vector is at the simulation boundary, then boundary data is used to complete the buffer. When the edge of the vector is in the middle of the domain, then the vector must cross the front. FTI_getCrossing() returns the coordinate of this crossing point. The front state (§4.2) at this coordinate having the same component is used for all buffer states beyond the crossing. For an illustration of this construction see Fig. 4.5.

4.2.3 Support Functions

In addition to these modifications, we require the client to provide several callback functions for the proper execution of the front tracking dynamic geometry. We also break down these listed tasks into smaller functions. Here is the complete list of functions that the client must implement.

- Level Set Function (for initialization)
- Get Front State
- Front State Interpolation
- MOC Solver (dynamic state only)
- Tangential Solver (dynamic state only)

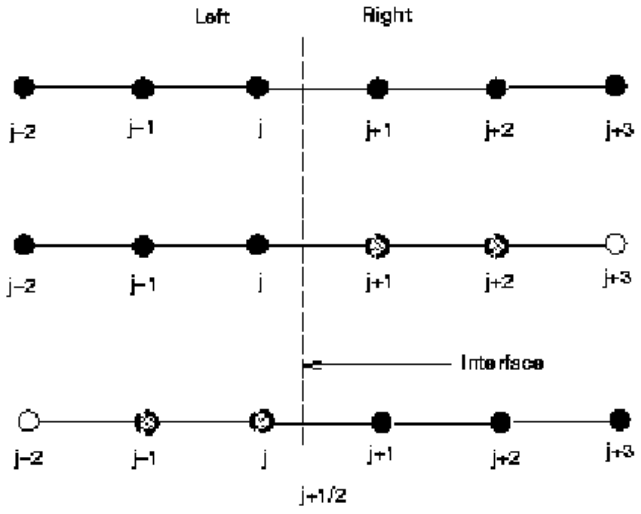

Figure 4.5: Ghost cell extrapolation. Above, original stencil, crossing the interface. Middle: Two right states are replaced by left side extrapolated states to define a stencil consisting of only left states. Below: a similar construction for the right sided stencil.

4.2.3.1 Level Set Function

The level set function is given as the function $\phi : \mathcal{R}^n \to \mathcal{R}$ which maps each point in the spatial domain to its (signed) distance from the front. The sign distinguishes between the two regions.

For example, to create an initial interface which is a sphere of radius 1 centered at the origin with positive component inside and negative component outside, we define:

$$\phi(x, y, z) = 1 - (x^2 + y^2 + z^2)\,. \tag{4.6}$$

Each front separates two regions. To initialize many fronts, a level set function is used to initialize each one individually. Allowing these fronts to intersect is not supported here.

4.2.3.2 Get Front State

The FTI server provides the client routine with a front point and its normal. An outline of the front state algorithm follows:

1. Get front point coordinates

2. Get normal

3. Obtain two outer front states
    - Component aware interpolation
4. Solve Riemann problem between outer front states
5. Return mid-states from Riemann Solution

The resulting mid states are the required front states, each corresponding to one of the interior components. This algorithm is used to initialize the dynamic-state model, and it is used for all front states in the state-free model.

4.2.3.3 Component Aware Interpolation

Component aware interpolation is used to compute outer front states in several of the algorithms, ensuring that each outer front state averages only cell values that match the component of one side of the front.

We implement component aware interpolation on the dual grid to the simulation, where components are defined on grid cell corners. See §4.1.2. To find the state at an arbitrary point, we identify the cell on the dual grid which contains this point. The algorithm constructs an interpolation element using only grid data from a single component taking the first to succeed of the four following constructions.

In 2D:	In 3D:
1. Bilinear Interpolation	1. Trilinear Interpolation
2. Linear Interpolation (inside)	2. Bilinear Interpolation (inside)
3. Linear Extrapolation (outside)	3. Bilinear Extrapolation (outside)
4. Constant Extrapolation	4. Constant Extrapolation

The inputs to this function are the coordinates to be used for state evaluation, and the component consistent with those coordinates. We have implemented this function by constructing a dual grid to the simulation grid (see 4.1.2). Each corner of the dual grid is a cell center of the original grid associated with an interior state and a component identifier.

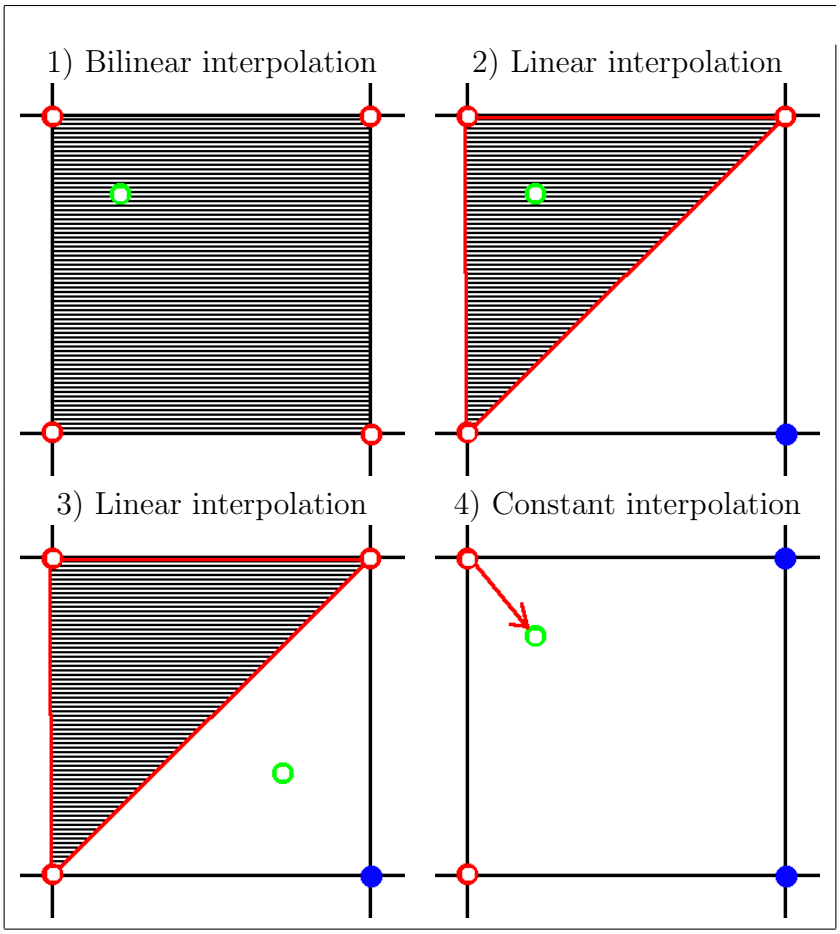

Figure 4.6: Cases of the component aware interpolation model in 2D. The shaded area shows the interpolation or extrapolation element. Case 1 shows a bilinear interpolation case. Case 2 shows linear a interpolation case. Case 3 shows linear a extrapolation case. Case 4 shows a constant extrapolation case.

We first determine the dual grid cell in which the point lies, and from this we perform a walk over the corners associated with that dual grid cell. Each cell corner state that matches the component chosen is added to a queue storing its coordinates and state. Depending on how many points are available, different different types of interpolation are used. See Figs. 4.6 and 4.7 for the details of each construction.

4.2.3.4 Riemann Solver

In order to compute the front state, a Riemann problem is solved in the direction normal to the front. We use an exact Riemann solver [5].

We first take S_1 and S_2 from component aware interpolation at the given coordinates using component 1 and component 2 respectively. Using [5] we have two functions for mid

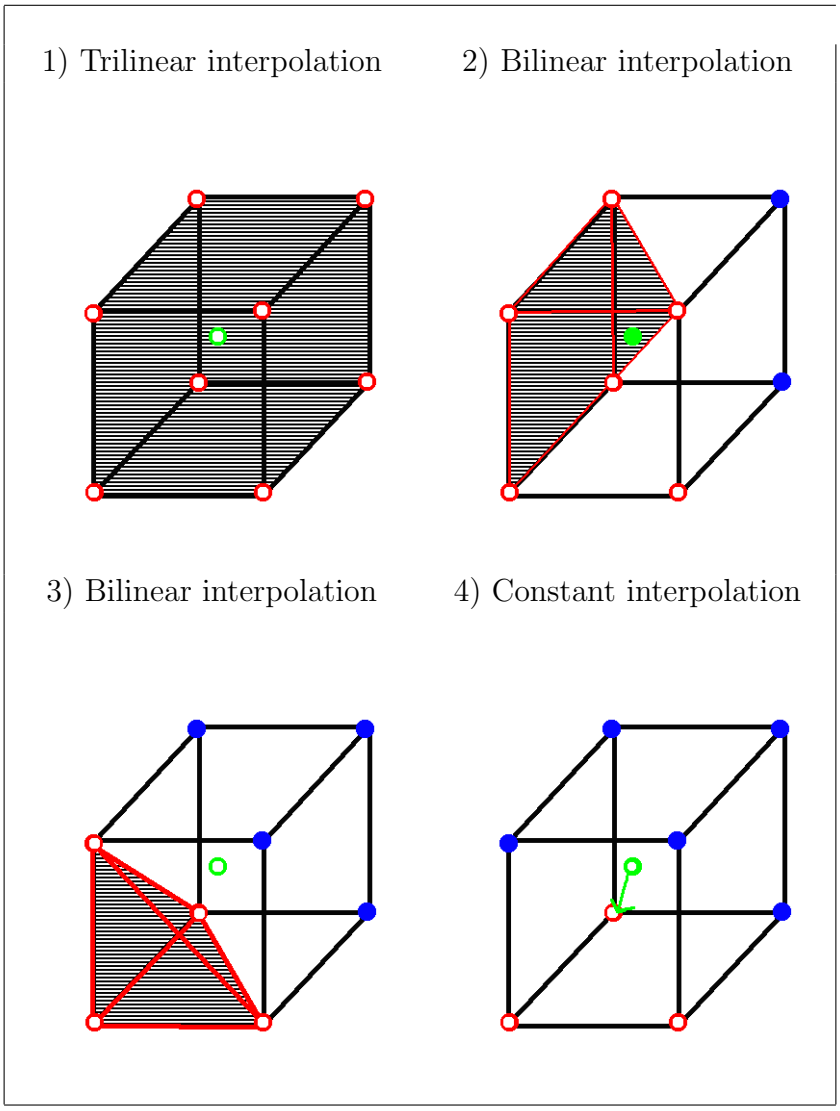

Figure 4.7: Cases of the component aware interpolation model in 3D. The shaded area shows the interpolation or extrapolation element. Case 1 shows a trilinear interpolation case. Case 2 shows bilinear a interpolation case. Case 3 shows bilinear a extrapolation case. Case 4 shows a constant extrapolation case.

state velocity, one taking data from S_1 across a left facing wave, and one taking data from S_2 across a right wave. These problems are underdetermined and have a free parameter, taken as the pressure. We denote these $u_l(S_1, p_l^*)$ and $u_r(S_2, p_r^*)$, where p_l^* and p_r^* are the mid state pressures. Since pressure and velocity are continuous across the contact wave, we have:

$$p_l^* = p_r^* = p^* \tag{4.7}$$

$$u_l(S_1, p^*) = u_r(S_2, p^*) \tag{4.8}$$

This is solved by a numerical root finder. This gives pressure and velocity in the mid state. Density and energy are calculated using wave equations in [5] and the EOS.

4.2.3.5 Front State Interpolation

This method is used in dynamic-state mode to fill in state values to dynamically created points when the front is remeshed. It is a simple interpolator which takes in 3 sets of coordinate / front state pairs and a set of target coordinates. It returns the linearly interpolated results. This method uses the client state structure which is opaque to the FTI server. Hence it must be implemented by the client.

4.2.3.6 MOC Solver

This routine is used in dynamic-state mode to propagate the front state in the normal direction along with its motion. For the Euler equations , there are three space-time characteristics (§4.1.4). When the front is advanced, the point is moved according to the normal velocity of its front states. Since these are mid states from a Riemann problem (with continuous normal velocity), both states have the same normal velocity. The front point is a contact discontinuity, and thus travels along the C^0 characteristic. The front state information will travel along the C^+ and C^- characteristics, which can be traced back to a point interior to the solution grid at t_n. This information (which is typically not a grid point) must be interoplated (via Component Aware Interpolation: §4.2.3.3), and then taken to define the new front state at t_{n+1}.

We can perform this algorithmically as:

$$p^{(1)} = p(0) + v^{(0)} \cdot \delta t$$

$$s_l^{(1)} = \text{interp(coords, compl)}$$

$$s_r^{(1)} = \text{interp(coords, compr)}$$

These states are used as outer states in a Riemann problem and the resulting mid states define the new front states after the normal propagation.

4.2.3.7 Tangential Solver

As described in §4.1.7.2.2, the front is flattened around a single point. This is done by projecting the front onto the tangential plane. This creates an unstructured state grid surrounding a single point. This point is updated by the flux generated by a solve on these states. Since finite difference solvers rely on grid structure, we interpolate this unstructured mesh onto a stencil compatible with the interior solver. The interior solver is used to calculate the flux using this stencil, which is added into the solution state at that point. This is iterated over all front points.

4.3 FTI Server Implementation

4.3.1 Server Role in Front Tracking Algorithm

- Initialize (uses client callback functions)
  - Set up initial geometry
  - Initialize front states
- Time Step (uses client callback functions)
  - Front propagation
  - Resolve topology changes
  - Remeshing and smoothing
- Support Functions
  - calculate normal and curvature
  - determine component region
  - determine if front crosses mesh line
  - compute front state at crossing of front with mesh line

4.3.2 Time Step Advancement

It is the role of the server to automate the propagation of front points and states. This task is performed using client functions for physics related tasks, and can be divided into steps.

1. Normal Propagation: points move, and states are updated via MOC (dynamic-state only)
2. Tangential Propagation: tangential update to dynamically tracked front states (dynamic-state only)
3. Resolve Topology Changes: a LGB untangle algorithm is used to resolve changing topology.
4. Remeshing and Smoothing: small elements are joined, and large elements are split to maintain quality of the Lagrangian front mesh.

During the normal propagation, we use client functions to obtain the contact velocity at the front point and to update the front state at the point via the MOC solver (§4.2.3.6). This is done at each point to construct a second order RK solution to Eqn. 4.5.

The tangential propagate step constructs a tangential stencil to the front at each front point. That regular stencil is populated with states interpolated from nearby front states and passed into the interior solver for a flux calculation.

The Riemann problem is solved again on the new front states to guarantee continuous pressure and normal velocity.

Topological changes are resolved by adding, deleting, and/or joining elements near the location where the front becomes tangled (geometrically inconsistent). This is a sophisticated geometric algorithm [9] which FronTier is capable of performing internally, without support from the client. However, new front points require new states to be calculated via the client front state interpolation algorithm (§4.2.3.5).

Remeshing and smoothing are done to maintain mesh quality [46]. Again, this operation is entirely geometric, and thus handled by the server unaided except for computing new states at added points by front state interpolation as with the changing topology step.

4.3.3 Support Functions

4.3.3.1 Normal and Curvature

The FTI server can compute normal and curvature at the front. The front point normal is computed and stored during the front point propagation step. These normals can be

interpolated to any point on the front surface by giving input coordinates.

4.3.3.2 Component

This method is crucial to the front tracking algorithm. Given a set of coordinates, the FTI server can return the component, an indicator of the region in which the coordinates lie. This is needed for component aware interpolation, determination of irregular solution stencils, and for ghost cell extrapolation. It is used in the marching front to fill in the components onto the client grid (§4.1.2).

4.3.3.3 Mesh Line Crossing

Given the coordinates of a line segment, the FTI server can determine if that segment crosses the front, and return the coordinates of the crossings if they exist.

4.3.3.4 Front State at Mesh Line Crossing

Given the coordinates of a line segment, the FTI server will return the front state at the crossing point of this segment with the front. This is intended to get the solution at the point where a cut stencil crosses the front (where the line represents an arm of the stencil).

4.4 FTI Implementations

Implementing the physics routines outlined allows a simple and robust front propagation for a wide variety of potential clients. Implementing the ghost fluid method, a restructuring of the basic solution algorithm, avoids numerical diffusion across the front by avoiding solution stencils which contain data from different materials.

FTI is currently being used in the SBU cFluid and iFluid (compressible and incompressible fluid) packages, the high energy density physics code FLASH, and (with passive tracking only) by the weather simulation code WRF (see Table 4.4).

FTI is designed to be lightweight, simple, and easy to use and understand. Detailed outlines of these definitions follow in the FTI manual attached in Appendix .

4.4.1 FTI Server: FronTier

FronTier is a front tracking simulation code developed and maintained at SUNY Stony Brook. FronTier supports triangulated surface meshes and a dynamic propagation algorithm

Table 4.1: FTI Client Implementations

Code	Description	Status	Location
FLASH	High Energy Density Physics	In progress	This work
cFluid	SBU fluid code	In progress	This work
iFluid	SBU fluid code	Future work	SBU
WRF	Weather forecast	Passive tracking	SBU thesis
Petroleum Reservoir	Flame fronts	Planned	IMPA
Electrocardiac		In progress	SBU thesis

which supports changing topology. FronTier also performs the normal and curvature calculations necessary to perform front tracking related calculations, as well as component query to determine (quickly) the named region (component) of a specified set of coordinates.

Geometry based server functions based on the front come from a recent reformulation of the interface package, performed by Xiangmin Jiao and students [17, 18, 39]. The idea is to construct a coordinate neighbor patch (stencil) about each front point of sufficient size to allow geometry operations (e.g. normals, curvature) to be computed with higher order accuracy. A 1-ring about a point consists of all triangles with the point as a vertex. A $1\frac{1}{2}$-ring includes also all triangles with an edge in common with a 1-ring triangle. Larger stencils are built by continuation of this idea. On a stencil, the surface is represented as a polynomial height function of fixed order, leading to linear equations for the polynomial coefficients for the local height function which defines the surface. The stencil is chosen to be larger than needed to solve these equations, which are thus overdetermined. They are then solved (approximately) by least squares. The purpose of the least squares construction is to achieve a robust algorithm.

Geometrical operations (normal, curvature) have a straight forward derivation within this framework, and more general integral and differential surface operations are also supported.

4.4.2 Clients

4.4.2.1 cFluid

We provide a reference implementation of the FTI client in the Stony Brook CFD code cFluid. cFluid and iFluid are compressible and incompressible object oriented CFD codes developed by Xiaolin Li based on the FronTier simulation package at Stony Brook.

4.4.2.2 FLASH

FLASH is a high energy density physics code developed at uChicago. It supports PPM and WENO, MHD unsplit staggered mesh, and MHD split 8 wave solvers. FLASH also has

a variety of EOS available. There are also models for laser energy deposit, opacity, particle methods, nuclear burning, and more. Its modular design makes it highly extensible. This implementation will add front tracking to the list of available modules in FLASH.

4.4.2.3 WRF

The Weather Research and Forecasting (WRF) Model is a mesoscale numerical weather prediction system. The model serves a wide range of meteorological applications across scales from tens of meters to thousands of kilometers. WRF is currently in operational use at NCEP, AFWA, and other centers. WRF features the nonhydrostatic mesoscale model (NMM) dynamic solver, including one-way and two-way static nesting. WRF also features the unified post processor (UPP) software package and sample scripts for several graphical packages.

WRF stores meteorological variables at discrete points on a mesh. We intend to use the tracked surfaces from FTI to represent cloud boundaries, determined by data from satellite images. This is being developed to make short term predictions for cloud coverage over solar farms. Future applications may be to track uncertainty boundaries around weather patterns such as storms.

4.5 Future Work

4.5.1 Second Order Conservative Tracking

The FT API is based on the currently active vision of FronTier. Here we outline a second order conservative algorithm, based on [30, 46]. In fact, higher than second order accuracy follows from a higher order version of the ideas presented here. The ideas follow from [16].

1. Propagate the front to a full time level and also to a 1/2 time level
2. Use the space-time front to divide the space-time cells it crosses into cut cells, i.e. two subregions, each on one side of the front.
3. Merge cut cells with small tops (at time level $n+1$) or even lacking tops, with neighbors, until all merged cells have a top volume $\approx \Delta x \Delta y \Delta z$.
4. Integrate the flux $\nabla F \cdot n$ over the side or time-like cut-cell boundaries and also over all regular side and cut-cell faces of the space-time cut cell.
5. The solution can be evaluated on each side wall of the cut cell by differencing to the required order in standard numerical methods.

6. Conservative differencing gives the solution average over the top (level $n+1$) in terms of the average over the bottom (level n) plus the sum of the side fluxes.

7. Integration of the flux over the $t = n + \frac{1}{2}$ cut cell surface is supported in the software package, and is sufficient for a second order algorithm. Extensions to higher order follow similar ideas.

This plan is a modification of the algorithm of [14, 30]. The main improvement is in step (7) which reduces the cost of the flux integration, so that the algorithm is feasible to implement. Additionally, tetrahedralization of the cut cell side surface is not required.

This version of FT eliminates the ghost cell extrapolation and the interpolation algorithm for cell centers crossing (in time) the moving front.

In contrast to [30], there is no use of an explicitly discretized space-time front. That is, step 2 is conceptual only and does not occur in the algorithm. Step 4 is carried out (for example) at the $n + \frac{1}{2}$ time level, using the $n + \frac{1}{2}$ time front (see step 7). The side faces of the space-time cell can be either regular or cut. For cut cell side faces, we evaluate the flux at the centroid, to achieve second order accuracy. Thus, we never deal with the geometry of a 3D space-time surface in 4D space-time. In this regard, the present algorithm improves over [16].

4.5.2 Embedded Boundary Method

The embedded boundary method (EBM) is a numerical algorithm for the solution of elliptic problems with discontinuous data and/or source terms. We refer to this method here as it is consistent with the ideas of front tracking, and is needed for use of front tracking for elliptic problems. The method only uses data from cell centers and the geometry of the discontinuous surface. We consider an elliptic equation of the form

$$\nabla \cdot F = f \tag{4.9}$$

$$F = A \nabla U \,, \tag{4.10}$$

where A and f may be discontinuous over a surface and U is the unknown. The method was introduced by Colella, and further improved in [44, 1]. The idea is to introduce new degrees of freedom in the solution matrix at the centroids of the cut cells, and to evaluate flux F and source f terms at centroids of cut cell faces. These evaluations proceed from the cut centroid data.

4.5.3 Late Time Selective Untracking

The front motion is largely grid free. This subgrid resolution offers many advantages. However a problem sometimes arises in late time turbulent flow simulations using front tracking in which unphysically small regions may be tracked. Also late-time mixing can cause the regions near the front to become so diffuse, that the tracked interface is no longer physical.

The idea is, according to some criteria, we wish to declare a triangle to be untracked or passive. This property of the triangle must be returned via the stencil / front intersection query. When the crossing point belongs to an untracked triangle, the ghost fluid method will not be applied. Instead the cell will update normally. [16]

4.5.4 Other Physics

4.5.4.1 Flame Fronts

We can extend the methods outlined here to apply front tracking to tracking flame fronts in a combustion simulation. These flame fronts are structured with chemical reactions taking place in the interior and on the boundary. This has been studied in [36] which notes that the thermonuclear fusion fronts in type IA supernova behave similarly to combustion flame fronts. Tracking flame fronts poses new and unique challenges. The flame speed is determined by a source term integrated over the flame front.

4.5.4.2 Magnetohydrodynamics

Coupling the dynamics of moving surfaces into MHD has required some additional development. The hyperbolic dynamics are the same as the fluid applications here. However, MHD has an elliptic component as well. The embedded boundary method, as above, must be implemented for the MHD equations. Some techniques have been developed and validated in [41, 8] using FronTier, but are not yet part of the FTI code.

4.5.4.3 Elasticity

Modeling elastic sliding surfaces is another potential application of front tracking and FTI. This is done by a master-slave model. Master and slave nodes are created at the slide surface, and shear forces are computed following an ODE. Front tracking offers promise for extending the geometric capabilities of such a model, as well as standardizing its implementation into a fundamental package compatible with many other models.

Wstar Manual

CHAPTER

ONE

USER'S GUIDE

1.1 Introduction

Wstar is a tool for computing W* convergence of CFD, and other simulation data. The main idea is a grid coarsening and sampling. The coarsened grid contains in each cell, a sample of data generated from the simulation grid. This sample regarded without any geometry coming from the point locations within the supercell. It is instead considered to be a sample of probability density function(PDF) in the coarse grid cell on the space of solution state variables. This PDF is computed in the supercell and serves as the Young's measure for w* convergence.

There are two steps involved in the w* convergence test. The first is the generation of PDF data from the coarsen and sample model. The second is the comparison of two simulations by comparing these coarsened grids of Young's measures.

The first step, coarsen and sample, is performed by the driver (reference?)wstar.py. The second step, compare is perforemed by the driver (reference?)wstar_compare.py

1.2 Algorithm Details

The coarsen and sample step is fairly straight forward. A new, coarse grid is constructed concurrently with the simulation data grid. Iterating over the simulation data, the values are binned and counted in the corresponding coarse cell. After this is done, the simulation data is thrown away, and PDF's are generated based on the bin frequencies in the coarse cells. The bins are considered to be a piecewise constant approximation to the true PDF.

The comparison has some more detail. First, the data ranges on the two data to compare may not have the same binning or values. The 'bin-lines', grid lines of the discretized PDF, are taken in union to make a new binning or grid (perhaps irregular) for the PDF's. The data are extrapolated as piecewise constant onto the new grid for each PDF.

Now, the PDF's may be integrated into CDF's. This is done by a Riemann integral. This integral is exact (numerically, not stastically) because the bins are piecewise constant, and that constant is chosen to give the correct integral over the bin to reflect the frequencies of the data sample. Whereas the the PDF has values associated with bins, the CDF has values associated with the grid lines (the first value being always 0, the last being always 1) and is interpreted as a piecewise linear approximation to the true CDF.

The comparison for PDF and for CDF are slightly different.

1.3 Wstar Usage

This package contains several module files with .py extensions, as well as two executable drivers: wstar.py, and wstar_compare.py

Each is invoked in the same way, passing a single command line argument representing the input file from which to operate.

1.3.1 Sample wstar.py input file

```
{
        # This will match the file test8.DENSITY.scalar.ts00822-nd0000.vtk etc.
        InputName=test8.%var.scalar.ts%step05d-nd%node04d.vtk
        # Output file name will be ./pdf-test/test.vtk
        OutputName=pdf-test/test

        # Number of node files (cuncurrent domains)
        Nodes=64

        # Size of supercell in terms of simulation cells.
        nx=5
        ny=5
        nz=5
        nt=5

        # Data about which variables, timesteps, etc to operate upon
        BaseStep=822
        Steps=822,825,828,831,834
        Vars=DENSITY
        # Data ranges and binning size
        ValRange=0,1
        nBins=20
}
```

This is a sample input file for wstar.py.

InputName The name of the input file. Certain markups are recognized:

%noded This will substitute the node number into the file name. The final 'd' is as in the printf format string for integers. For example, if you have files like: name01.txt, name02.txt,... You can use: 'name%node02d.txt'

%stepd The subs for the time step number. This can be formatted like %noded.

%var The name of the variable being analyzed.

OutputName The name of the output file. '.pdf' will be appended to that name.

Nodes The number of node files (concurrent domains) to be read in at each time step. They will be numbered starting at 0.

nx,ny,nz The number of normal cells to a supercell in each dimension.

nt The number of timesteps to a supercell (time discretization).

BaseStep The base time step. This should be the first entry in the steps list

Steps The list of timesteps to be included in the analysis. This should be a comma separated list of integers. The first element should be the same as BaseStep. The number of entries should be the same value as nt.

Vars This should be a comma separated list of variables to be cross correlated e.g. var0,var1,... Often only one variable is used, but multiple variable correlations can be done.

ValRange The range of values in the input. This is necessary to perform the PDF binning before processing can begin. This is a comma separated list with entries being min0,max0,min1,max1,... with each pair corresponding to an entry in the Vars list.

nBins The number of bins in the PDF discretization. This is a comma separated list with each entry corresponding to an entry in the Vars list.

The complete data is set in curly braces. Several braced blocks will run several PDF generations on different blocks of data (from the same simulation or from different simulations). The order of the keys is not important. Each key must appear exactly once in a block, or an exception will be raised.

1.3.2 Sample wstar_compare.py input file

```
{
        File1=/gpfs/scratch2/yanyu/test_mirko_110902/pdftest/pdf-test.CONCENTRATION.pdf.ts00822
        File2=/gpfs/scratch2/yanyu/test_mirko_110902/pdftest/pdf-test.CONCENTRATION.pdf.ts01693
        OutputFile=./pdftest/concentration_comparison
        Integrate=yes
}
```

This is a sample input file for wstar_compare.py.

File1,File2 The two PDF files upon which to perform comparison. These should be the output of a prior execution of wstar.py.

OutputFile The name of the output file. '.vtk' will be appended to the given file path and name.

Integrate Should be 'yes' or 'no'. Yes indicates that the PDF will be integrated into CDF before comparison. 'no' will perform the comparison on the raw PDF.

1.3.3 Command line invocation

Wstar can be invoked at the command line. This is done by calling:

```
wstar.py input_file_name
```

The output will be a single file per block in the input file given by the OutputName key (with '.pdf' appended). This file will contain all of the PDF data for performing comparisons.

TODO: document the file format, perhaps in an appendix. TODO: provide binary output

1.3.4 Running unit tests

The test directory contains some testing protocols for wstar. The 'dotests.py' file will execute the tests and report the number of successes and failures. All detailed output will be piped into the log file, 'test.log'.

CHAPTER

TWO

DEVELOPER'S GUIDE

This is the developer's guide.

2.1 Coding style

2.1.1 File organization

Wstar is coded in Python in an object oriented style. It is divided into several files:

wstar.py The driver for computing PDF's from simulation data

wstar_compare.py The driver for comparing grids of PDF's generated by the above

pdf.py All of the code pertaining to PDF's and CDF's directly. This includes objects which generate PDF's from data, integrating PDF's to CDF's, and comparison functions for the PDF's/CDF's

grid.py This contains all of the data structures of the grids used. This includes simulation grids, PDF grids, CDF grids, and tools for performing wstar comparison of grids (successively calling compare functions in the PDF module).

fgrid.py This module provides a few factory functions for GRID based on different data formats for simulation data.

arraynd.py Since correlations can be on any number of variables, this module provides an n-dimensional array object (arraynd) that can be indexed by a list of indices. It also provides some tools for iterating over, and manipulating such index lists.

utils.py Some global utilities. Included are the input file parser, and some floating point tolerance utilities.

2.1.2 Coding Practices

- It is preferred to use tab=4 spaces. All indentation is done by tabs.
- Data members prefixed by a double underscore are private, and should not be accessed outside of the constructor. These data are immutable and should only be accessed by the properties assigned to them.
- Function members prefixed by double underscore are private. They may be used outside of the constructor, but only by function members of the owning object.
- Data/function members prefixed by a single underscore are protected. They are meant to be accessed by the owning class and by subclasses.
- Variable/members are sad camels if they are local variables, proud camels if they are intended for some broader use.

2.2 Module Documentation

2.2.1 wstar

wstar.py: This is the wstar main driver for generating PDF grids from simulation data.

It has the capability of generating several pdf grids from several simulations, however the file reader is currently only capable of reading in one pdf grid to make.

The main routine invokes the FileDriver, which parses the input file into the necessary data. That data is stored in a STEP object. Several step objects can be consolidated into a step list. This list is then passed on to the MainDriver.run() which generates the PDF data.

class wstar.**FileDriver**(*fileName*)

This is a derived class from MainDriver. It takes an input file and generates the data necessary to call MainDriver. See the sample input file for more info

Parameters **fileName** (*string*) – The name of the input file to read

exception wstar.**GridCompatibilityError**(*filename*, *attribute*)

An exception class for compatibility errors between grids in the same timestep cluster. This will be raised when a file is read in that contains a grid not consistent with the previously read grids.

class wstar.**InputFileName**(*string*)

There are various naming schemes for files. This object is a file name generator for the filenames of the data input to wstar. Once instantiated, it will generate a file name based in the input:

node step var

get(*step*, *node*, *var*)

Return the generated filename for the given arguments :param step: The time step number :type step: int :param node: The id of the node when reading parallel data :type node: int :param var: The name of the variable to be read. :type var: string

class wstar.**MainDriver**(*steps*)

The main driver class. This class takes in the necessary information to do some analysis. The arguments are all of the required parameters. This class is the parent of FileDriver which extracts these parameters from the input file and passes them in.

Parameters **steps** (list(STEP)) – a list of analysis parameters

run(*steps*)

The model for this function is: looping over each variable, then for each major timestep, then for each node domain: construct a grid of supercells initialized to zero. then looping over each minor timestep in the time stencil: add the contents of each grid cell to its corresponding supercell

class wstar.**STEP**(*InputNameGen*, *OutputName*, *Base_step*, *Dim*, *Steps*, *Vars*, *Nodes*, *valRange*, *nBins*)

A single PDF grid to make. Tells which variable, timesteps to use

Parameters

- **InputNameGen** (InputFileName) – A filename generator for the input file.
- **OutputName** (*string*) – the name of the output file
- **Base_step** (*int*) – The first time step in the time step cluster
- **Dim** (*int*) – The grid dimension
- **Steps** (*list(int)*) – A list of all time step numbers in the cluster These should be in order, and the first entry should be equal to Base_step

- **Vars** (*list(string)*) – A list of all variables to be analyzed. The order of these matters to the next several arguments.
- **Nodes** (*int*) – the number of nodes (domain files)
- **valRange** (*list(tuple(int))*) – A list of 2-tuples who specify the extreme ranges (min,max) of each variable in Vars in corresponding order
- **nBins** (*list(int)*) – An array of integers representing the bin discretization of the corresponding variable in vars

exception `wstar.`**`StepConsistencyError`**(*message*)
An exception class for inconsistent input data for analysis. This may be raised during initialization when the input file is processed for consistency.

2.2.2 wstar_compare

wstar_compare.py: Performs PDF/CDF comparison analysis for W* convergence tests. Reads in PDF files generated by wstar.py.

exception `wstar_compare.`**`ComparisonError`**(*grid1*, *grid2*, *attr*)
To compare two grids, some attributes must match If they do not, this exception is raised.

class `wstar_compare.`**`PDF_COMPARE`**(*pdf_grid1*, *pdf_grid2*, *outname*)
This is a class that runs a comparison on two pdf grids. The final analysis is written to outname

Parameters

- **pdf_grid1** (`PDF_GRID`) – the first pdf grid.
- **pdf_grid2** (`PDF_GRID`) – the first pdf grid.
- **outname** (*string*) – the output file name for final analysis

class `wstar_compare.`**`PDF_COMPARISON`**(*inFileName1*, *inFileName2*, *outFileName*, *integrate*)
The parameters for a PDF comparison. Takes the filenames of the two PDF files (generated by wstar.py), the name of an output file, and a flag indicating whether or not to integrate to CDF before doing the comparison

Parameters

- **basename1** (*string*) – the first pdf file
- **basename2** (*string*) – the second pdf file
- **outbasename** (*string*) – the output file name

class `wstar_compare.`**`PDF_COMPARISON_DRIVER`**(*inputfilename*)
The driver to run the wstar comparison. This integrates the pdf grids based on the input option, and then calls `PDF_COMPARE` to do the comparison

Parameters **inputfilename** – the name of the input file for the comparison

`run`(*comparisons*)
Do the indicated analyses. Loop over list of comparisons to perfoem. For each: read in the PDF file, integrate if indicated, then call `PDF_COMPARE` to generate the analysis.

Parameters **comparisons** (list(`PDF_COMPARE`)) – The list of analyses to perform

2.2.3 grid

class `grid.`**`GRID`**(*gmax*, *spacing=None*, *origin=None*, *varName=None*, *initialval=0.0*)
A grid class for multi purpose use. gmax is a 3-tuple giving the grid spacing in each dimension

static **combine_grids**(*grids*)
A factory function to make a grid from several grids

writeVTK(*filename*)
Write the data to VTK format works only if grid is of singe value floats

class grid.**PDF_GRID**(*filename*)
This is the grid of PDF's used in wstar_compare.py. It is related to the SUPERGRID.

compare(*pdf_grid1*, *pdf_grid2*)
Compare two pdfgrids.

integrate()
Call the integrate method on each of the IRREGULAR_PDF's in the grid. This will replace each IRREGULAR_PDF with an IRREGULAR_CDF object.

unify_binning(*pdf1*, *pdf2*)
This function takes 2 pdf grids and replaces all PDF objects with IRREGULAR_PDF objects on the same grids.

class grid.**PDF_MASTER**(*pdfgrids*)
This class takes in several pdf_grid's and constructs a master pdf_grid from them. It uses the origin field to arrange the grids passed in into a grid of grids called the "domain_grid". It is assumed that they can fit together to form a rectangular grid of domains. Then each domain in the domain_grid is assigned a startindex. This represents the number of indices below that grid in each dimension, and is used to shift the indices when constructing the master grid of all of the domains.

construct_global_grid(*domain_grid*)
Take entries in the grids in the domain grid and place them into the global grid at the correct global indices.

static **get_domain_grid_dimensions**(*pdfgrids*)
Find the dimensions of the domain grid. This is equal to the partition dimeneions. This is done assuming the grid array, pdfgrids, is in z,y,x sorted order. It works by comparing the origins to see how many there are in each diimension There is some reduncency in the calculation for the purpose of error catching.

static **get_total_grid_dimension**(*domain_grid*)
Go through each of the grids in the domain grid and compute the total overall grid dimensions

static **set_pdfgrid_start_index**(*domain_grid*)
Take in the domain grid, and set the value: startindex on each pdfgrid in it. This is the shift from local/domain coords to global coords used to construct the master grid.

static **sort_grids**(*pdfgrids*)
Sort the grids in order of z then y then x. This way they are in row major form to be made into a 3d array of grids.

class grid.**SUPERGRID**(*gmax*, *var*, *origin*, *spacing*, *valRange*, *nBins*)
The grid of superblocks. input determines the corresponding grid, and the number of grid cells to a supercell. It is assumed to divide evenly

Finalize()
Analogous to the finalize method of PDF_BUILDER() class. This calls the PDF_BUILDER.Finalize() on each PDF in the supergrid. The result is that the supergrid now contains cells with normalized PDF's with calculated first and second moments.

2.2.4 fgrid

2.2.5 pdf

class pdf.**IRREGULAR_CDF**(*array*, *lines*, *nBins*, *valRange*)

This is a CDF that does not have a regular grid spacing It is similar to the IRREGULAR_PDF, and is generated by that class's integrate() method. It contains all of the data of CDF, with the addition of some knowledge of the physical coordinates of the grid lines.

class CELL(*x*, *f*, *bilin_eval=None*)

A bilinear cell in n dimensions. x is array of x coords [(x0min,x0max),(x1min,x1max),...] f is ARRAYND of function values. Dim in each var should be 2.

analytic_integral()

Perform integration of bilinear cell analytically.

bilinear_evaluate_generator(*xa*)

Evaluate the bilinear interpolant on the cell at the point xa

integrate()

Integrate the cell, automatically determining if analytic or numerical integration is required.

numerical_integral()

Split the cell by FACTOR in each dimension, and do a riemann integral.

numerical_integral2()

If there is a crossing, split the cell by 2 in each dimension. continue to split recursively until there is no cross, or the cell size is very small.

IRREGULAR_CDF.**compare**(*cdf1*, *cdf2*)

Perform the W* comparison on cdf's. Return the integrated difference scaled by the total area (scale is on [0,1]).

IRREGULAR_CDF.**get_cell**(*index*)

Given the cell index, return a billinear cell object (IRREGULAR_CDF.CELL).

IRREGULAR_CDF.**integrate**()

Get the integral of the given CDF, intended to be a function in CDF data structure, but representing the difference of 2 CDF's. The integration is performed analytically on cells where there is no detected crossing of the bilinear functions (same signs at all grid points), but numerical in the presence of a crossing.

Iterates over all bilinear cells and finds the integral.

class pdf.**IRREGULAR_PDF**(*pdf*, *lines*)

This is a PDF object whose grid lines are not evenly spaced. The initialization is from a regularly spaced pdf, and a list of grid lines. The new pdf is created using constant interoplation on the PDF bins at the given grid lines.

compare(*pdf1*, *pdf2*)

Do the W* comparison on two PDF's. Integrated difference times area. For PDF's, this is on a scale [0,2], and so the result is multiplied by 1/2 to get a scale of [0,1].

integrate()

Generate a CDF from the PDF. This is done by riemann integrating. Riemann is the proper integeral to use, due to the piecewise constant definition of the PDF.

class pdf.**PDF**(*pdfarray*, *nBins*, *valRange*, *mean*, *cov*)

This object represents a PDF. It stores the PDF as a multi-dimensional array in state-variable space. The elements in the array represent probability density for the discretized bin which it represents.

Parameters

- **pdfarray** (*list(float)*) – The pdf in array form.
- **valRange** (*list(2-tuple(float))*) – The ranges of values for each variable

:param nBins : the number of bins for each variable :type nBins : list(int) :param mean : The mean of the PDF :param cov : the covariances of the PDF

There is correntlly little to no use of the mean and cov, but they may prove useful in the future for analysis.

tostr()
Return a string representation of the PDF for printing, and for future reading by the PDF_FROMSTR object.

class pdf.**PDF_BUILDER**(*Vars*, *nBins*, *valRange*)
The Probability density function object. It stores a PDF as an arrayND(above) Representing bins in the PDF. Each bin has an integer containing the abs frequency of the range of the bin. Bin ranges are computed arithmetically.

When complete, the getPDF() method converts the integer counts to probability densities and generates a PDF object.

Parameters

- **Vars** –
- **valRange** –
- **nBins** –

addValue(*value*)
Add a value to the PDF. The correct bin is computed arithmetically and the freq for that bin is increased.

getPDF()
Computes mean/std/etc shifts from absolute frequency(int) to relative freq (float on [0,1]) returns a PDF object

class pdf.**PDF_FROMSTR**(*string*)
A class derived from PDF whose only initializer is a string. The string converts directly into a complete pdf. the string is exactly the output of a PDF.tostr() call.

PYTHON MODULE INDEX

INDEX

U

W

FTI Manual

0.1 File Index

0.1.1 File List

Here is a list of all documented files with brief descriptions:

0.2 File Documentation

0.2.1 FTI_client.h File Reference

FTI Client Functions.

Functions

- void FTI_getVelocity (const double *coords, double *return_vel)
- void FTI_getFrontStates (double *coords, int component, State *return_st)
- void FTI_ghostBuffer (clientStencil *stencil)
- void FTI_Interpolate (double *coords, STATE *return_state)
- void FTI_RiemannSolver (ClientState leftOuter, ClientState rightOuter, ClientState return_leftInner, ClientState return_rightInner)
- void FTI_NormalUpdate (ClientState old_state, double *coords_old, double *coords_new, ClientState *new_state)
- void FTI_TangentialUpdate (ClientStencil stencil)

0.2.1.1 Detailed Description

FTI Client Functions. This is the list of FTI client functions. These functions handle the physics and grid related aspects of front tracking.

0.2.1.2 Function Documentation

0.2.1.2.1 void FTI_getVelocity (const double * *coords*, double * *return_vel*)

Input the coordinates, return the velocity

0.2.1.2.2 void FTI_getFrontStates (double * *coords*, int *component*, State * *return_st*)

Input the coordinates and normal, return two front states.

Parameters

coords	the coordinates of the front point at which to find front state
return_vel	the returned velocity

0.2.1.2.3 void FTI_ghostBuffer (clientStencil * *stencil*)

Given the stencil, compute the ghost states and fill them in.

0.2.1.2.4 void FTI_Interpolate (double * *coords,* STATE * *return_state*)

Given the coordinates and component, interpolate the state.

0.2.1.2.5 void FTI_RiemannSolver (ClientState *leftOuter,* ClientState *rightOuter,* ClientState *return_leftInner,* ClientState *return_rightInner*)

Given the coordinates and component, interpolate the state.

0.2.1.2.6 void FTI_NormalUpdate (ClientState *old_state,* double * *coords_old,* double * *coords_new,* ClientState * *new_state*)

Given the coordinates of a point during its update, find the new front state by method of characteristics.

0.2.1.2.7 void FTI_TangentialUpdate (ClientStencil *stencil*)

Given the coordinates of a point during its update, find the new front state by method of characteristics.

0.2.2 FTI_server.h File Reference

FTI Server Functions.

Enumerations

- enum FT_BOUNDARY_TYPE { **FT_BOUNDARY_PERIODIC**, **FT_BOUNDARY_REFLECTING**, **FT_BOUNDARY_FLOW** }
- enum FT_GEOMETRY_TYPE { **FT_GEOMETRY_SPHERICAL**, **FT_GEOMETRY_CARTESIAN**, **FT_GEOMETRY_CYLINDRICAL**, **FT_GEOMETRY_POLAR** }

Functions

- void FTI_Init_Dynamic_State (int *dim, int *procGrid, double *xmin, double *xmax, double *ymin, double *ymax, double *zmin, double *zmax, int *isize, int *jsize, int *ksize, int *ibuf, int *jbuf, int *kbuf, int *xlbdry, int *xubdry, int *ylbdry, int *yubdry, int *zlbdry, int *zubdry, double(*level_func)(void *func_params, double *coords), void *level_func_params, int(*vel_func)(void *, double *, double *), void *vel_params, int *geometry)
- void FTI_Init_Dynamic_State (int *dim, int *procGrid, double *xmin, double *xmax, double *ymin, double *ymax, double *zmin, double *zmax, int *isize, int *jsize, int *ksize, int *ibuf, int *jbuf, int *kbuf, int *xlbdry, int *xubdry, int *ylbdry, int *yubdry, int *zlbdry, int *zubdry, double(*level_func)(void *func_params, double *coords), void *level_func_params, int(*vel_func)(void *, double *, double *), void *vel_params, int *geometry void *tangential_update(ClientStencil stencil);void *normal_update(ClientState old_state, double *coords_old, double *coords_new, ClientState *new_state);)
- void FTI_Output (const char *filename, int length)
- void FTI_WriteRestart (const char *filename, int length)
- void FTI_Propagate (const double *dt)
- void FTI_CreateSurface (int pos_comp, int neg_comp, double(*level_func)(void *, double *), void *level_params, int surf_type, int(*vel_func)(void *, double *, double *), void *vel_params)
- int FTI_getComponent (double *coords)
- void FTI_normal (int *size, double **coords, double **nor)
- void FTI_cross (const double **line, double *cross)
- void FTI_state_at_cross (const double **line, double *cross, ClientState *lstate, ClientState *rstate)

0.2.2.1 Detailed Description

FTI Server Functions. This is the list of FTI server functions. These functions handle the geometric aspects of front tracking, and provide hooks for client level and velocity functions for initialization and time step advancement of the front.

0.2.2.2 Enumeration Type Documentation

0.2.2.2.1 enum FT_BOUNDARY_TYPE

Boundary type

0.2.2.2.2 enum FT_GEOMETRY_TYPE

Geometry type

0.2.2.3 Function Documentation

0.2.2.3.1 void FTI_Init_Dynamic_State (int * *dim*, int * *procGrid*, double * *xmin*, double * *xmax*, double * *ymin*, double * *ymax*, double * *zmin*, double * *zmax*, int * *isize*, int * *jsize*, int * *ksize*, int * *ibuf*, int * *jbuf*, int * *kbuf*, int * *xlbdry*, int * *xubdry*, int * *ylbdry*, int * *yubdry*, int * *zlbdry*, int * *zubdry*, double(*)(void *func_params, double *coords) *level_func*, void * *level_func_params*, int(*)(void *, double *, double *) *vel_func*, void * *vel_params*, int * *geometry*)

The FTI Initialization function

Parameters

dim	the dimension of the simulation. Supported 2D or 3D.
xmin	lower x coordinate bound of interior domain
xmax	upper x coordinate bound of interior domain
ymin	lower y coordinate bound of interior domain
ymax	upper y coordinate bound of interior domain
zmin	lower z coordinate bound of interior domain
zmax	upper z coordinate bound of interior domain
isize	number of interior grid cells in x direction
jsize	number of interior grid cells in y direction
ksize	number of interior grid cells in z direction Does lots of initialization related stuff

0.2.2.3.2 void FTI_Init_Dynamic_State (int * *dim*, int * *procGrid*, double * *xmin*, double * *xmax*, double * *ymin*, double * *ymax*, double * *zmin*, double * *zmax*, int * *isize*, int * *jsize*, int * *ksize*, int * *ibuf*, int * *jbuf*, int * *kbuf*, int * *xlbdry*, int * *xubdry*, int * *ylbdry*, int * *yubdry*, int * *zlbdry*, int * *zubdry*, double(*)(void *func_params, double *coords) *level_func*, void * *level_func_params*, int(*)(void *, double *, double *) *vel_func*, void * *vel_params*, int *geometry void *tangential_update(ClientStencil stencil);void *normal_update(ClientState old_state, double *coords_old, double *coords_new, ClientState *new_state);)

The FTI Initialization function

Parameters

dim	the dimension of the simulation. Supported 2D or 3D.
xmin	lower x coordinate bound of interior domain
xmax	upper x coordinate bound of interior domain
ymin	lower y coordinate bound of interior domain
ymax	upper y coordinate bound of interior domain
zmin	lower z coordinate bound of interior domain
zmax	upper z coordinate bound of interior domain
isize	number of interior grid cells in x direction
jsize	number of interior grid cells in y direction
ksize	number of interior grid cells in z direction Does lots of initialization related stuff

0.2.2.3.3 void FTI_Output (const char * *filename*, int *length*)

Write out visualization data to a file.

Generates visualization data for the front. The output format is VTK POLYDATA.

Parameters

filename	The name of the file to generate
length	The length of the filename string (for FORTRAN compatibility)

0.2.2.3.4 void FTI_WriteRestart (const char * *filename,* int *length*)

Write out restart data to a file.

Generates a restart file for the front.

Parameters

filename	The name of the file to generate
length	The length of the filename string (for FORTRAN compatibility)

0.2.2.3.5 void FTI_Propagate (const double * *dt*)

This is the time step function for the front.

Advance the front by the time step Δt. Move the points according to the velocity field.

Parameters

dt	The time step size Δt

0.2.2.3.6 void FTI_CreateSurface (int *pos_comp,* int *neg_comp,* double(*)(void *, double *) *level_func,* void * *level_params,* int *surf_type,* int(*)(void *, double *, double *) *vel_func,* void * *vel_params*)

Create a single surface.

Parameters

pos_comp	the component identifier where the level function is positive
neg_comp	the component identifier where the level function is negative
level_func	a function pointer for a level function describing the surface
surf_type	a client controlled identifier for the surface. It can have any integer value. Surfaces of the same type will will have the same velocity function.
vel_func	a velocity callback function for the surface.
vel_params	some client controlled data that can be passed into the velocity function.

0.2.2.3.7 int FTI_getComponent (double * *coords*)

Get the component at the specified coordinates

Parameters

coords	The coordinates

Returns

The component

0.2.2.3.8 void FTI_normal (int * *size,* double ** *coords,* double ** *nor*)

Get the list of front coordinates and normals

Parameters

size	The number of points in the array
coords	the array of coordinates.
nor	the array of normals.

0.2.2.3.9 void FTI_cross (const double ** *line*, double * *cross*)

Get the intersection point of a line segment with the front

Parameters

line	coordinates of 2 points defining the line segment
cross	returned crossing coordinates

0.2.2.3.10 void FTI_state_at_cross (const double ** *line*, double * *cross*, ClientState * *lstate*, ClientState * *rstate*)

Get the front state at an intersection point of a line segment with the front

Parameters

line	coordinates of 2 points defining the line segment
cross	returned crossing coordinates
lstate	returned left front state
rstate	returned right front state

Index

Bibliography

[1] The embedded boundary method for two phase incomressible flow. *arXiv*, Arxiv.org ¿ math ¿ 1304.4414, 2013.

[2] J. Ball. A version of the fundamental theorem of Young measures. In *PDEs and Contunuum models of Phase Transitions, Lecture Notes in Physics*, volume 344, pages 207–215, New York, 1989. Springer Verlag.

[3] S. Chandrasekhar. *Hydrodynamic and Hydromagnetic Stability*. Dover Publications, New York, 1981.

[4] G.-Q. Chen. Convergence of the Lax-Friedrichs scheme for isentropic gas dynamics III. *Acta Mathematica Scienitia*, 6:75–120, 1986.

[5] A. Chorin and J. Marsden. *A Mathematical Introduction to Fluid Mechnics*. Springer Verlag, New York–Heidelberg–Berlin, 2000.

[6] G. Dimonte, D. L. Youngs, A. Dimits, S. Weber, M. Marinak, S. Wunsch, C. Garsi, A. Robinson, M. Andrews, P. Ramaprabhu, A. C. Calder, B. Fryxell, J. Bielle, L. Dursi, P. MacNiece, K. Olson, P. Ricker, R. Rosner, F. Timmes, H. Tubo, Y.-N. Young, and M. Zingale. A comparative study of the turbulent Rayleigh-Taylor instability using high-resolution three-dimensional numerical simulations: The Alpha-Group collaboration. *Phys. Fluids*, 16:1668–1693, 2004.

[7] X. Ding, G.-Q. Chen, and P. Luo. Convergence of the Lax-Friedrichs scheme for isentropic gas dynamics I and II. *Acta Mathematica Scientia*, 5:415–432, 433–472, 1985.

[8] J. Du, T. Lu, and R. Samulyak. Algorithms for magnetohydrodynamics of ablated materials. *Journal of Nanoscience and Nanotechnology*, 8:3674–3685, 2008.

[9] Jian Du, Brian Fix, James Glimm, Xicheng Jia, Xiaolin Li, Yunhua Li, and Lingling Wu. A simple package for front tracking. *J. Comput. Phys.*, 213:613–628, 2006.

[10] W. Gangbo and M. Westerdickenberg. Optimal transport for the system of isentropic Euler equations. *Comm. PDEs*, 34:1041–1073, 2009.

[11] J. Glimm and X. L. Li. Recent progress in turbulent mixing. In M. Legrand and M. Vandenboomgarde, editors, *Proceedings of the 10th International Workshop on the Physics of Compressible Turbulent Mixing, 17-21 July, Paris France*, 2006. Stony Brook University Preprint SUNYSB-AMS-06-13, In Press.

[12] J. Glimm and D. H. Sharp. Simulation vs. theory vs. experiment for complex fluid mixing flows. *SIAM News*, 2006. Stony Brook University Preprint SUNYSB-AMS-06-05.

[13] J. Glimm, D. Marchesin, and O. McCryan. Statistical fluid dynamics: Unstable fingers. *Comm. Math. Phys.*, 74:1–13, 1980.

[14] J. Glimm, X.-L. Li, Y.-J. Liu, Z. L. Xu, and N. Zhao. Conservative front tracking with improved accuracy. *SIAMJNA*, 41:1926–1947, 2003.

[15] J. Glimm, D. H. Sharp, T. Kaman, and H. Lim. New directions for Rayleigh-Taylor mixing. *Phil. Trans. R. Soc. A*, 371:20120183, 2013. doi: http://dx.doi.org/10.1098/rsta.2012.0183. Los Alamos National Laboratory Preprint LA-UR 11-00423 and Stony Brook University Preprint SUNYSB-AMS-11-01.

[16] J. Glimm, H. Lim, R. Kaufman, and W. Hu. Euler equation existence, nonuniqueness and mesh converged statistics. 2014. Stony Brook University Preprint SUNYSB-AMS-14-04.

[17] Xiangmin Jiao. Face offsetting: a unified framework for explicit moving interfaces. *J. Comput. Phys.*, 220:612–625, 2007.

[18] Xiangmin Jiao and Duo Wang. Reconstructing high-order surfaces for meshing. *Engineering with Computers*, 28:361–373, 2012.

[19] T. Kaman. *Rayleigh-Taylor Turbulent Mixing Simulations.* Ph.d. thesis, Stony Brook University, 2012.

[20] T. Kaman, J. Glimm, and D. H. Sharp. Uncertainty quantification for turbulent mixing simulation. *5th International Conference of Numerical Modeling of Space Plasma Flows (ASTRONUM 2010)*, 444:21, 2010. Stony Brook University Preprint number SUNYSB-AMS-10-04. Los Alamos National Laboratory preprint LA-UR 11-00422.

[21] T. Kaman, R. Kaufman, J. Glimm, and D. H. Sharp. Uncertainty quantification for turbulent mixing flows: Rayleigh-Taylor instability. In A. Dienstfrey and R. Boisvert, editors, *Uncertainty Quantification in Scientific Computing*, volume 377 of *IFIP Advances in Information and Communication Technology*, pages 212–225. Springer, 2012. Stony Brook University Preprint number SUNYSB-AMS-11-08.

[22] Ryan Kaufman, Tulin Kaman, Yan Yu, and James Glimm. Stochastic convergence and the software tool W*. In *Proceeding Book of International Conference to honour Professor E.F. Toro*, pages 37–41. CRC, Taylor and Francis Group, 2012. Stony Brook University Preprint number SUNYSB-AMS-11-10.

[23] A. N. Kolmogorov. Local structure of turbulence in incompressible viscous fluid for very large Reynolds number. *Doklady Akad. Nauk. SSSR*, 30:299–3031, 1941.

[24] H. Lee, H. Jin, Y. Yu, and J. Glimm. On validation of turbulent mixing simulations of Rayleigh-Taylor mixing. *Phys. Fluids*, 20:1–8, 2008. Stony Brook University Preprint SUNYSB-AMS-07-03.

[25] D.K. Lilly. A proposed modification of the germano subgrid-scale closure method. *Physics of Fluids*, 4:633–636, 1992.

[26] H. Lim, J. Iwerks, J. Glimm, and D. H. Sharp. Nonideal Rayleigh-Taylor mixing. *Proc. Nat. Acad. Sci.*, 107(29):12786–12792, 2010. Stony Brook University Preprint SUNYSB-AMS-09-05 and Los Alamos National Laboratory Preprint LA-UR 09-06333.

[27] H. Lim, J. Iwerks, Y. Yu, J. Glimm, and D. H. Sharp. Verification and validation of a method for the simulation of turbulent mixing. *Physica Scripta*, T142:014014, 2010. Stony Brook University Preprint SUNYSB-AMS-09-07 and Los Alamos National Laboratory Preprint LA-UR 09-07240.

[28] H. Lim, Y. Yu, J. Glimm, X. L. Li, and D. H. Sharp. Subgrid models for mass and thermal diffusion in turbulent mixing. *Physica Scripta*, T142:014062, 2010. Stony Brook Preprint SUNYSB-AMS-08-07 and Los Alamos National Laboratory Preprint LA-UR 08-07725.

[29] H. Lim, T. Kaman, Y. Yu, V. Mahadeo, Y. Xu, H. Zhang, J. Glimm, S. Dutta, D. H. Sharp, and B. Plohr. A mathematical theory for LES convergence. *Acta Mathematica Scientia*, 32:237–258, 2012. Stony Brook University Preprint SUNYSB-AMS-11-07 and Los Alamos National Laboratory Preprint LA-UR 11-05862.

[30] J. Liu. *A Conservative Front Tracking Method in N-Dimensions*. PhD thesis, Stony Brook Univ., 2006.

[31] X.-F. Liu, E. George, W. Bo, and J. Glimm. Turbulent mixing with physical mass diffusion. *Phys. Rev. E*, 73:056301, 2006.

[32] T. Ma. *Large eddy simulation of variable density flows.* PhD thesis, University of Maryland, 2006.

[33] J. Melvin, P. Rao, R. Kaufman, H. Lim, Y. Yu, J. Glimm, and D. H. Sharp. Atomic scale mixing for inertial confinement fusion associated hydro instabilities. *High Energy Density Physics*, 9:288–298, 2013. Stony Brook University Preprint SUNYSB-AMS-12-01 and Los Alamos National Laboratory Preprint LA-UR 12-21555.

[34] N. Mueschke and O. Schilling. Investigation of Rayleigh-Taylor turbulence and mixing using direct numerical simulation with experimentally measured initial conditions. I. Comparison to experimental data. *Physics of Fluids*, 21:014106 1–19, 2009.

[35] Nicholas Mueschke and Oleg Schilling. Investigation of Rayleigh-Taylor turbulence and mixing using direct numerical simulation with experimentally measured initial conditions. II. Dynamics of transitional flow and mixing statistics. *Physics of Fluids*, 21: 014107 1–16, 2009.

[36] J. C. Niemeyer, W. K. Bushe, and G. R. Ruetsch. *Approaches to modeling thermonuclear flames.* Center for Turbulence Research, Proceedings of the Summer Program, Stanford, CA, 1998.

[37] R. Di Perna. Convergence of approximate solutions to conservation laws. *Arch. Rational Mech. Anal.*, 82:27–70, 1983.

[38] R. Di Perna. Compensated compactness and general systems of conservation laws. *Trans. Amer. Math. Soc.*, 292:383–420, 1985.

[39] N. Ray, D. Wang, X. Jiao, and J. Glimm. High-order numerical integration over discrete surfaces. *SIAM Journal Numerical Analysis*, 50:3061–3083, 2012.

[40] R. D. Richtmyer. Taylor instability in shock acceleration of compressible fluids. *Comm. pure Appl. Math*, 13(297-319), 1960.

[41] R. Samulyak, J. Du, J. Glimm, and Z. Xu. A numerical algorithm for MHD of free surface flows at low magnetic Reynolds numbers. *J. Comput. Phys.*, 226:1532–1546, 2007.

[42] J. Smagorinsky. General circulation experiments with the primitive equations. *Mon. Weather Rev.*, 91:99–165, 1963.

[43] V. S. Smeeton and D. L. Youngs. Experimental investigation of turbulent mixing by Rayleigh-Taylor instability (part 3). AWE Report Number 0 35/87, 1987.

[44] Shuqiang Wang, Roman V Samulyak, and Tongfei Guo. An embedded boundary method for elliptic and parabolic problems with interfaces and application to multi-material systems with phase transitions. *Acta Mathematica Scientia*, 30B(2):499–521, 2010.

[45] A. Yoshizawa. Statistical theory for compressible turbulent shear flows, with the application to subgrid modeling. *Physics of Fluids*, 29(2152-2164), July 1986.

[46] Yijie Zhou. PhD thesis, Stonybrook University.