A Blurred Interface Formulation of The Reference Map Technique for Fluid-Solid Interactions and Fluid-Solid-Solid Interactions

by

Boris Ivanov Valkov

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY June 2014

© Massachusetts Institute of Technology 2014. All rights reserved.

Signature redacted

Author .........................................................

Department of Mechanical Engineering
March 19, 2014

Signature redacted

Certified by ..............................................

Kenneth Kamrin
Assistant Professor of Mechanical Engineering
Thesis Supervisor

Signature redacted

Accepted by ....................................................

David E. Hardt
Chairman, Department Committee on Graduate Students
A Blurred Interface Formulation of The Reference Map Technique for Fluid-Solid Interactions and Fluid-Solid-Solid Interactions

by

Boris Ivanov Valkov

Submitted to the Department of Mechanical Engineering on March 19, 2014, in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Abstract

In this work we present a blurred interface method for Fluid-Solid Interactions (FSI) and multiple solids immersed in a fluid or FSSI (Fluid-Solid-Solid Interactions) based on the reference map technique as presented by Kamrin and Rycroft [12]. I will follow the chain of thought which lead from the initial sharp interface technique to the newer blurred interface one. We will present its capabilities of doing fully-coupled simulations of a compressible Navier-Stokes fluid and highly non-linear solid undergoing large deformations all performed on a single Eulerian grid with no Lagrangian particles whatsoever.

The Reference Map Technique (RMT) provides an Eulerian simulation framework allowing to compute fully coupled fluid/soft-solid interactions [12]. However, due to the extrapolations inherent to the Ghost Fluid Method [7] (GFM) for fluid/fluid interactions, on which the RMT is based, numerical artifacts get created in the resulting pressure and velocity fields whenever the levelset defining the interface crosses a grid-point from the fixed cartesian grid utilized in this method. We will therefore follow the creation and propagation of these artifacts as well as analyze how the blurred technique solves or avoids these problems.

Thesis Supervisor: Kenneth Kamrin
Title: Assistant Professor of Mechanical Engineering
Acknowledgments

I would like to thank my advisor Ken Kamrin for inviting me to join his team and giving me the opportunity to work on an exciting, new and innovative problem which can potentially redefine how we think about various mechanical problems. Furthermore, I am grateful to him for his high-level of investment in closely following the project and assisting me in every step both from a conceptual and technical point of view, as well as, for his thrust in me and giving me the liberty to be the driving force of my project.

My journey to graduate studies has never been rectilinear and I would like to thank all of the people who supported me all the way from Sofia to Paris and then to Boston. The professors at Lycée Chaptal in Paris gave me a once in a lifetime opportunity and warm welcome in my first year abroad. Professor Jacques Fontanier and the whole outstanding teaching staff at Lycée Le Corbusier in Aubervilliers then believed in my capabilities and managed not only to prepare me for the competitive entrance examinations but also managed to give me back the spark for mathematics and engineering, and make me gain the self-confidence which lead me to Arts et Métiers ParisTech. From then on, Louis Magne, all of the staff of Arts et Métiers, as well as the Li210, made sure that I excel in all possible ways leading me to MIT. Once here, I had the chance to learn from and interact with remarkable professors, friends and colleagues both in classes and in the Kamrin Group. My housemates Jacques and Daniel were a great team. The support of the graduate office and in particular Ms. Leslie Regan was crucial for moving on. The financial support given to me through the Shapiro and Martin fellowships gave me the unique chance to attend MIT.

Last but not least, I could not have done any step of this journey without the infinite support of my loving family Slavka, Ivan, Tatyana and my love Nina all of whom have shared with me every success and every difficulty.

За Сео Алевра и гого Борем.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.3</td>
<td>The new stencils for rmap gradient and stress divergence</td>
<td>53</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Mixing quantities across the interface</td>
<td>55</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Outline of the method</td>
<td>56</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Reduced number of extrapolation steps</td>
<td>58</td>
</tr>
<tr>
<td>4.4</td>
<td>The importance of the reference map extrapolation</td>
<td>59</td>
</tr>
<tr>
<td>4.5</td>
<td>Discrepancies between the interface locus as per RMAP and as per levelset</td>
<td>60</td>
</tr>
<tr>
<td>4.6</td>
<td>Improved pinning technique</td>
<td>63</td>
</tr>
<tr>
<td>4.7</td>
<td>Density mixing and density update</td>
<td>64</td>
</tr>
<tr>
<td>4.8</td>
<td>Applying diffusion to the extrapolated values of the reference map</td>
<td>65</td>
</tr>
<tr>
<td>4.9</td>
<td>Results</td>
<td>67</td>
</tr>
<tr>
<td>4.10</td>
<td>Convergence</td>
<td>71</td>
</tr>
<tr>
<td>4.11</td>
<td>Simulation of contact between two solids immersed in a fluid</td>
<td>77</td>
</tr>
<tr>
<td>4.11.1</td>
<td>Formulation</td>
<td>78</td>
</tr>
<tr>
<td>4.11.2</td>
<td>Setting contact conditions</td>
<td>81</td>
</tr>
<tr>
<td>4.11.3</td>
<td>Results</td>
<td>84</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions</td>
<td>89</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary</td>
<td>89</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
<td>90</td>
</tr>
<tr>
<td>A</td>
<td>The corrected extrapolation routine for matching boundary conditions</td>
<td>93</td>
</tr>
<tr>
<td>A.1</td>
<td>Motivation</td>
<td>93</td>
</tr>
<tr>
<td>A.2</td>
<td>The extrapolation correction routine ensuring boundary condition at the interface</td>
<td>94</td>
</tr>
<tr>
<td>B</td>
<td>Removing fake surface tension forces on curved boundaries</td>
<td>99</td>
</tr>
<tr>
<td>C</td>
<td>Rmapsim.cc source code</td>
<td>103</td>
</tr>
<tr>
<td>D</td>
<td>Rmaptest.cc source code</td>
<td>133</td>
</tr>
</tbody>
</table>
E Rmapsimtypes.cc source code

135
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Motion function</td>
<td>20</td>
</tr>
<tr>
<td>2-2</td>
<td>Reference map function</td>
<td>20</td>
</tr>
<tr>
<td>2-3</td>
<td>Storage of quantities on a regular and staggered grid</td>
<td>23</td>
</tr>
<tr>
<td>2-4</td>
<td>Outline of the Reference map technique</td>
<td>23</td>
</tr>
<tr>
<td>2-5</td>
<td>The computational domain of our simulations: regular grid (white), staggered grid (red crosses), interface (white curve) defining a solid region (green), a fluid region (blue) and a partial solid/fluid region (orange)</td>
<td>24</td>
</tr>
<tr>
<td>3-1</td>
<td>Example of simulations carried out with the initial formulation of the sharp interface implementation of the RMT</td>
<td>26</td>
</tr>
<tr>
<td>3-2</td>
<td>Example of a “blip” appearing at the interface between the fluid and the solid</td>
<td>27</td>
</tr>
<tr>
<td>3-3</td>
<td>Example of wrinkles appearing on the interface when no surface tension is applied</td>
<td>28</td>
</tr>
<tr>
<td>3-4</td>
<td>A wrinkle growing on the interface until it reaches the neighboring gridcell when it snaps back to its original position</td>
<td>29</td>
</tr>
<tr>
<td>3-5</td>
<td>Mixing of quantities across the interface based on the fraction of fluid/solid in the partial grid cell</td>
<td>30</td>
</tr>
<tr>
<td>3-6</td>
<td>The X component of the reference map together with its extrapolation in the case of the stick simulation</td>
<td>33</td>
</tr>
<tr>
<td>3-7</td>
<td>Stress obtained from extrapolated reference map even before the solid undergoes any deformation as in a stress free state</td>
<td>34</td>
</tr>
</tbody>
</table>
4-5 The computational domain of our simulations: regular grid (white), staggered grid (red crosses), interface (white curve) defining a solid region (green), a fluid region (blue) and a partial solid/fluid region (orange) .......................................................... 53

4-6 Old and new stencils for calculation of divergence of stress .............. 54

4-7 Old and new stencils for calculation of the gradient of reference map 54

4-8 The zero contour of the levelset as well the $\phi$ function ................. 55

4-9 The definition of the error function, the adapted error function and the adapted error function around the interface in a 2D problem ............... 56

4-10 The solid $11$ stress as obtained with the first implementation of the blurred interface method .......................................................... 61

4-11 The stress obtained from extrapolated reference map showing correlation with the wrinkles forming on the interface .................. 62

4-12 Drift between the interface as described by the levelset (white contour) and as described by the reference map (blue contour) .............. 63

4-13 Verification of the diffusion algorithm making sure that we obtain a smooth extrapolation of a field ........................................... 67

4-14 Verification of the diffusion algorithm making sure that we obtain a smooth extrapolation of a field ........................................... 68

4-15 Stress field $\sigma_{11}^{cf}$ from extrapolated reference map without diffusion showing striations ......................................................... 69

4-16 Stress field $\sigma_{11}^{cf}$ from extrapolated reference map with diffusion resulting in a smooth extrapolation .................................... 70

4-17 Stress field $\sigma_{11}^{cf}$ from extrapolated reference map without diffusion showing a sudden ripple in stress due to a jump in extrapolated values 71

4-18 Stress field $\sigma_{11}^{cf}$ from extrapolated reference map with diffusion not showing any ripples .................................................... 72

4-19 Resulting pressure field simulation of a stick anchored at one end getting deformed to large strain by the surrounding fluid flow ............ 73
4-20 Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid .............................................. 74
4-21 Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid .............................................. 75
4-22 Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid .............................................. 76
4-23 Convergence plot for horizontal velocity component field $u_f$ for grid-sizes 60x60, 72x72, 90x90, 120x120, and 180x180 compared against a simulation with grid-size 360x360. .............................................. 77
4-24 Pressure field at frame t=40 for the rotor simulation with a grid of 60x60, 72x72, 90x90 and 120x120 grid-points. ................................................................. 78
4-25 Pressure field at frame t=40 for the rotor simulation with a grid of 180x180 and 360x360 grid-points. ................................................................. 79
4-26 Zoomed in comparison in pressure fields at the boundary with a grid of 60x60 points versus a grid of 360x360 points. ................................................................. 79
4-27 Convergence plot for horizontal reference map component field $\xi_x$ for grid-sizes 60x60, 120x120, and 180x180 compared against a simulation with grid-size 360x360. .............................................. 80
4-28 Convergence plot for horizontal reference map component derivative field $\frac{\partial \xi_x}{\partial x}$ for grid-sizes 60x60, 120x120, and 180x180 compared against a simulation with grid-size 360x360. .............................................. 81
4-29 Example of erroneous penetration of the solid phases when the contact condition has not been explicitly set ................................................................. 82
4-30 The difference function $\phi_{12}$ between the two levelsets giving us the mid-surface between the two solids ................................................................. 83
4-31 Zoom in on the Gaussian function (colored contours) centered around the mid-surface to correctly define contact and avoid penetration .............................................. 84
4-32 Example of contact between an anchored and a mobile ball showing large deformations under contact immersed in a fluid .................. 85

4-33 Simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid ............................. 86

4-34 Simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid (part two) .................... 87

4-35 Final frame of the simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid .............. 88

A-1 Mismatch of two extrapolated fields at the interface due to numerical imperfections .................................................. 95

A-2 Mismatch of two extrapolated fields at the interface due to numerical imperfections before and after the correction routine .............. 95

B-1 Comparison of divergence of extrapolated stresses on a straight and on a curved interface .................................................. 99

B-2 Calculating the fake body force created by extrapolating stress across a curved interface .................................................. 100
Chapter 1

Introduction

Simulations of fluid-solid interactions have long interested engineers and scientists. The fields of application are vast since one can imagine that in most practical situations solids are always immersed in some fluid. Starting from propellers, engines, and wings reacting with air or water and going to gear trains interacting with the lubricating liquid we always find a fluid-solid interface where the capability of modeling and predicting would greatly improve the design process.

However, in the world of numerical simulations there has always been a dichotomy between "fluid" methods and "solid" ones. From a constitutive point of view solid stress depends on total deformation and is computable from relative positions of neighboring nodes, whereas, fluid stress depends on the deformation rate and is therefore obtainable from the finite-difference of a velocity field on a fixed mesh [12]. It is therefore, easier to simulate a fluid in an Eulerian framework [5],[22],[23],[19],[9] and a solid in a Lagrangian one [25],[21],[10],[2] but the natural question which then arises is "How does one threat the fluid-solid interaction?".

Numerous approaches exist, all of which are based on different ideas. Some solve everything in a Lagrangian framework [6],[24],[17] but need to address problems of too large deformations of the mesh inside the fluid and therefore require a step of remeshing. Others, solve the fluid phase in an Eulerian frame, the solid one in a Lagrangian [6],[24] and then need to do complicated steps of projecting values from one to the other and vice versa.
The method we present here is a fully Eulerian method - both the fluid and the solid are solved for on the exact same fixed grid [13], furthermore, even the interface is tracked without the use of any Lagrangian pointers or particles whatsoever. As will be presented in the following document, this results in a method which is very simple from a philosophical point of view and is also easily implemented in an explicit time scheme.
Chapter 2

The Reference Map Technique for FSI

2.1 Introduction

The Reference Map Technique (RMT) is an explicit finite-difference method for finite-deformation solid laws [12] (Kamrin, Rycroft, Nave 2012). The key idea is the storage of a vector field called the reference map, which defines the origin of the points at the current location and allows the construction of the required kinematic quantities inside the solid phase.

People familiar with finite element methods (FEM) will recognize its similarity to the motion function which maps a point from the reference (undeformed) configuration to its position in the deformed one as illustrated in figure 2-1.

The reference map is actually the inverse concept - we map the points from the deformed configuration into the reference one as in 2-2.

2.2 Notation and Kinematics

As described by Kamrin et al. [12] in an Eulerian frame the conservation of mass and momentum in strong form (when deformations remain smooth) can be written
respectively as in 2.1 and 2.2.

\[ \rho_t = -v \cdot \nabla \rho - \rho \nabla \cdot v \]  \hspace{1cm} (2.1)

\[ v_t = -v \cdot \nabla v - (\nabla \cdot T + \rho g) / \rho \]  \hspace{1cm} (2.2)

Where \( \mathbf{v}(\mathbf{x},t) \) is the spatial velocity and \( \mathbf{T}(\mathbf{x},t) \) is the Cauchy stress tensor field. As previously discussed, we can express the motion of an element of material initially at \( \mathbf{X} \) moving to \( \mathbf{x} \) at time \( t \) as \( \mathbf{x} = \chi (\mathbf{X},t) \), therefore, defining the deformed configuration.
$B_t$. The motion then defines the deformation gradient $F$,

$$F(X, t) = \frac{\partial \chi(X, t)}{\partial X} \quad (2.3)$$

### 2.3 The reference map technique for general finite-strain elasticity

In our case, since the method is Eulerian we cannot use the motion function but instead use the reference map which we define as

$$X = \xi(x, t) \quad (2.4)$$

Using the chain rule it can then be shown that the deformation gradient $F$ is obtainable as in 2.5.

$$F(X, t) = (\nabla \xi(x, t))^{-1} \quad (2.5)$$

Once we have the deformation gradient we are capable of modeling any thermodynamically compatible solid law on a fixed grid. Throughout this work we have consistently used hyperelastic stress as simply

$$T = 2(\text{det} F)^{-1} F \frac{\partial \hat{\psi}(C)}{\partial C} F^T |_{F = (\nabla \xi)^{-1}, C = (\nabla \xi)^{-T} (\nabla \xi)^{-1}} \quad (2.6)$$

Here, $\psi$ is the Helmholtz free-energy per unit (undeformed) volume and $\cdot$ designates a function, therefore

$$\psi = \hat{\psi}(F) \quad (2.7)$$

The specific model applied in this work has always been the compressible neo-Hookean elastic solid model. The small-strain bulk and shear moduli are taken to be $\kappa = 10$ and $G = 10$ respectively. Letting $J = \text{det} F$ and indicating the deviator with $'$, we
define $\psi$ as in 2.8 and the Cauchy stress tensor is then given by equation 2.9.

\[
\psi = \frac{G}{2}(J^{-\frac{3}{2}}\text{tr}(B) - 3) + \frac{\kappa}{2}(J - 1)^2
\]  

(2.8)

\[
T_s = GJ^{-5/3}B' + \kappa(J - 1)I
\]  

(2.9)

For more information on general finite-strain elasticity refer and hyperelasticity refer to [8].

The full fluid stress tensor satisfies eq.2.10, where $\lambda_f$ is the fluid compressibility modulus, $\rho_0^f$ is the initial density and $\eta$ is fluid viscosity.

\[
T_f = \eta \frac{\nabla v_f + (\nabla v_f)^T}{2} - \lambda_f I - \frac{\rho_f^m}{\rho_0^f} - 1)I
\]  

(2.10)

We now also need an Eulerian update rule for the reference map $\xi$. We remind that the reference map tells us the origin of the current particle position, thus, we can observe that the reference map for a tracer particle never changes, giving us an advective evolution law

\[
\xi_t + \mathbf{v} \cdot \nabla \xi = 0
\]  

(2.11)

or in conservative form,

\[
(\rho \xi)_t + \nabla \cdot (\rho \xi \otimes \mathbf{v}) = 0.
\]  

(2.12)

In the case where the initial configuration is undeformed the initial condition for the reference map is just

\[
\xi(x, t = 0) = x = X.
\]  

(2.13)

The current implementation of the method is such that we utilize a regular, structured grid where we keep the kinematic quantities $\xi$ and $\mathbf{v}$ and a staggered grid with nodes in the cell centers of the regular grid where we keep the resulting fields at each time step - the deformation gradient $\mathbf{F}$ and the Cauchy stress tensor $\mathbf{T}$ (see fig. 2-3).

We now have all of the basic ingredients for the structure of the Reference map technique. Starting a given time step $n$ with given kinematic quantities (velocity and reference map) we solve for those quantities at time $n + 1$ as described in 2-4. The
Figure 2-3: Storage of quantities on a regular and staggered grid

The "basic" RMT routine; $Eu(\xi^n, v^n)$, strong formulation.
(All $\nabla$ operators discretize appropriately to centered or WENO finite-differences.)

<table>
<thead>
<tr>
<th>Given: $v^n$ and $\xi^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal: Calculate $v^{n+1}$ and $\xi^{n+1}$</td>
</tr>
<tr>
<td>Step 1: Construct $F$</td>
</tr>
<tr>
<td>Step 2: Compute $\rho$</td>
</tr>
<tr>
<td>Step 3: Compute $T$</td>
</tr>
<tr>
<td>Step 4: Update $v$</td>
</tr>
<tr>
<td>Step 5: Update $\xi$</td>
</tr>
</tbody>
</table>

$F = (\nabla \xi^n)^{-1}$
$\rho = \rho_0 (\text{det } F)^{-1}$
$T = T(F)$
$v^{n+1} - v^n + v^n \cdot \nabla v^n = \frac{1}{\rho} \nabla \cdot T + g$
$\xi^{n+1} - \xi^n + v^n \cdot \nabla \xi^n = 0$

Figure 2-4: Outline of the Reference map technique

computational domain we define consists of a regular and a staggered grid, a levelset function interpolated between gridpoints gives us the interface between phases and we can therefore define solid cells, fluid cells and partial ones as in fig. 2-5.
We construct:
- Regular grid
- Staggered grid (+)
- Level Set

We can define:
- A solid region
- A fluid region
- A partial solid/fluid region

Figure 2-5: The computational domain of our simulations: regular grid (white), staggered grid (red crosses), interface (white curve) defining a solid region (green), a fluid region (blue) and a partial solid/fluid region (orange)
Since the approach of our method is Eulerian, the grid on which information is saved never moves. This means that we need to define a way of distinguishing which points should be treated as fluid and which points should be treated as solid. The term sharp interface means that this distinction is binary - one point can either be one type or the other. The way we implement this numerically is by using a function $\phi(x)$ whose zero levelset defines the boundary of the solid, therefore, defining the interface between the solid and fluid phases [19],[16]. The levelset function lives on the regular gridpoints where the kinematic fields which move it are also kept. In between gridpoints $\phi$ can be interpolated using a bicubic interpolation scheme. It is then very simple to say that every regular gridpoint with a negative value of $\phi$ resides in the solid and every point with a positive value lies in the fluid. However, the question then arises on how to treat the grid cells through which the levelset crosses.
3.1 The occurrence of “Blips”

The sharp interface implementation of the Reference map method has been implemented and has been proved functional in various test cases such as an elastic rod anchored at the top getting deformed by a fluid flow, an elastic rotor anchored on a pivot in its center getting deformed up to large deformations and then spun by the fluid flow, as well as, a pre-strained disc which is released at time $t = 0$ creating pressure waves in the surrounding fluid (see fig. 3-1).

![Figure 3-1: Example of simulations carried out with the initial formulation of the sharp interface implementation of the RMT](image)

The proof of concept was fully made, however the simulation would sometimes get small glitches in the resulting fields right on the interface between the solid and the fluid and would result in the emission of a spherical wave, basically acting as a monopole source. These completely unphysical and random numerical artifacts are what we refer to as “blips” and in this chapter I will present the investigation carried out regarding the underlying reason for them.

As you can see in fig. 3-2 the blip in the top left-hand side of the solid appears as an artificial instantaneous pinching of the interface. We had various hypotheses about the possible reason for these artifacts but no full explanation of what was actually causing these problems. Our initial idea was that due to the various numerical extrapolations required in the sharp interface method, the extrapolated fluid velocities and the extrapolated solid velocities would not necessarily agree on the same value right at the interface. For a further discussion on this idea and a proposed solution see appendix A.
Figure 3.2: Example of a "blip" appearing at the interface between the fluid and the solid

### 3.2 Wrinkling of the interface

To obtain numerical stability in this formulation of our code we needed to use several "tricks" often applied in computational methods. One of them was that the solid actually had an artificial surface tension which keeps the interface together. In order to understand where the blips come from we needed to start turning off non-vital parts of the code to reveal the origin of the problem. The first thing we realized was that removing the surface tension actually allowed the interface to wrinkle whenever it through grid edges or gridpoints.

At this point it was very confusing why this type of motion of the interface is allowed since the underlying solid has an elastic response and therefore would not
accept its interface to have such defects but would naturally just “spring” back. Nevertheless, we zoomed in a particular region which was prone to wrinkling and started analyzing what was happening frame by frame looking at each grid cell. We then realized something very interesting - as the interface crosses through gridpoints it starts building up error and showing an instability in the form of wrinkles (fig.3-3). Afterwards, once the instability has occurred it starts growing until the moment it reaches the next gridcell at which time it is released and it then snaps back to where it should be thus causing the emission of a wave in both the fluid and the solid \(^1\) (fig.3-4). This seemed to be a perfect explanation of what the blips actually were with the only detail that we still did not know what caused the wrinkling in the first place.

### 3.3 Avoiding sudden “Identity changes”

Seeing how the interface snaps back when it reaches its neighboring gridcell we realized that during a certain period of time the partial cell through which the levelset was crossing did not realize it was solid and therefore did not apply the elastic response as described by the solid constitutive model. Eventually, when the levelset moves into

\(^1\)Throughout this chapter all of the figures regarding stress show the \(\sigma_{11}\) component as for the reasons of debugging there is no actual difference which component we analyze.
Figure 3-4: A wrinkle growing on the interface until it reaches the neighboring gridcell when it snaps back to its original position.

the next gridcell and out of the current one, the current one becomes what we would call a full-solid cell. At this point the cell would realize that its strain is huge and the solid constitutive law would create very high stress and pull back the interface. This process can be described as a sudden change of identity of a partial grid cell. It is very important at this moment to point out how different the point of view is when using an Eulerian method even just in terms of understanding what is actually happening in the simulation. In order to analyze the problem we should not anymore look at the interface and follow what it does while moving through gridpoints but we should look at a specific grid cell and see what it believes is actually happening. Putting one’s self in the place of a grid point or a grid cell was extremely useful and practical to allow us to play out in our minds what we are trying to instruct the method to do and what it actually does.

In other words, the life of a grid cell is as follows - the computational domain is divided into a fixed number of identical gridcells and all of them are born alike. At the very first time step some of the cells are called full-solids, some are called full-fluids and some get crossed by the interface, therefore, not truly certain whether to behave as a solid or a fluid. Initially, with the idea in mind to create a fully sharp interface
method this distinction was also kept sharp. The rule was such that the levelset gets interpolated at the center of the gridcell and if the obtained value is negative then the cell is more solid and therefore applies the solid constitutive law and vice versa if the value is positive the cell is more fluid than solid and therefore uses the fluid law. This makes a lot of sense when the picture is frozen but once you also consider time and the dynamics of the process you quickly realize that as the levelset moves through gridpoints they get a very sudden change in their constitutive law and thus in the stress calculated inside them.

To address this problem we came up with an approach which defines a gradual crossfade between the solid definition of stress and the fluid definition of stress based on the fraction of each of the phases in a particular grid cell. However, since this only carried out inside the partial grid cells this still remains a sharp interface technique. Another way to look at it would be to argue that the mixing of values only happens inside that one grid cell where we already only had an educated guess as to what stress should be and we simply give it the extra piece of information that a certain partial might be more or less fluid than another one.

Figure 3-5: Mixing of quantities across the interface based on the fraction of fluid/solid in the partial grid cell
One key advantage of the Eulerian method can be seen directly in this idea. The levelset function has a dual purpose - as well as tracking the interface, the value of the levelset anywhere in the computational domain is the distance between that point and the interface. We have therefore come up with a measure of the fraction of each phase inside a grid cell based on the $\phi$ values of its surrounding gridpoints as in 3.1.

$$\text{fraction} = f = \frac{1}{2}(1 + \frac{\sum_{i=1}^{4} \phi(i)}{|\sum_{i=1}^{4} \phi(i)|}) \quad (3.1)$$

Which then allows us to define the stress in a partial cell as being a mixture of the solid and fluid stress.

$$\sigma_{\text{partialgridcell}} = f \cdot \sigma_{\text{fluid}} + (1 - f) \cdot \sigma_{\text{solid}} \quad (3.2)$$

While applying this technique did not remove our wrinkles, or blips for that reason, it was definitely an improvement towards the stability and the smoothness of our simulations and it also set us on the right track to understanding what is actually happening around the interface. Taking the same logic even further we realized that we did not only have two possible formulations of stress inside a partial cell but we actually have four different ones:

- The solid stress obtained from direct extrapolation of full solid stresses
- The solid stress obtained from the solid constitutive law applied on the extrapolated reference map
- The fluid stress obtained from direct extrapolation of full fluid stresses
- The fluid stress obtained from the fluid constitutive law applied on the extrapolated fluid velocities

The first and simplest idea we then got was to actually define the stress in a partial grid cell as a mixture of all four of these. It is reasonable to think that as the levelset gets very close the solid the stresses obtained from extrapolated reference map should be more precise and as we get far away from the solid those which were extrapolated
directly from full solid stresses should be more reliable and similarly with their fluid equivalents. We experimented with this approach but it proved to be very unstable. However, this lead us to a crucial understanding of what the reference map method actually is and how we should approach the search for the origin of our instabilities. In order to see why this approach was unstable we decided to analyze our four ingredients in the stress mixture separately. A new conceptual idea was born at this time and it was to output what the stresses obtained from extrapolated reference map and to see what they look like not only in the partial grid cells but everywhere. This has been an essential concept throughout all of the work conducted on the reference map method ever since.

3.4 Understanding the importance of the reference map extrapolation

In the beginning we used to consider the extrapolation of the reference map as a technique which would allow us to have reference map value on both sides of the interface and thus allow us to solve the advection update rule for the reference map. However, looking into it more carefully we realize that the extrapolated reference map values at a specific location actually represent an initial guess for what the stresses would be if the solid was to then move to that location. Given that when a grid cell is full solid we obtain stress from the reference map this means that we create another type of "identity change" but this time between the extrapolated stress definition and the stress from extrapolated reference map.

Plotting this new field, to which we refer in short as stress from extrapolated reference map, we immediately observed a fundamental flaw in our code.

Even though, the reference map seems to be perfectly linear and correspond exactly to what we would expect it to be when extrapolated outside of the solid (fig.3-6), once we calculate the gradient as to obtain stresses we realize that there are flaws in the way the extrapolation routine creates new values in the presence of a curved
You immediately see that even our initial guess as to what stresses we should have in the solid if it were to move to the side just by a rigid motion actually has stress. The next thing one sees in this field is that the values are as expected when the interface is straight but completely wrong in the presence of curvature. Furthermore, as the simulation is carried on the solid starts to deform. This creates new curvature in the interface and introduces more of these numerical artifacts in the extrapolation (fig.3-8).

At this time we actually started to get a view of the big picture and started grasping how blips appear. In fig.3-9 we have a side-by-side comparison of the solid...
stress together with its extrapolation just before the levelset crosses a grid edge and right at the instant when it crosses. You can actually see a sudden jump in the extrapolated stress values which coincides perfectly with the same artifacts we see in the stress obtained from extrapolated reference map in the same region (fig.3-8).

We have now followed the mechanism of creation and propagation of a blip on the interface. In other words:

1. The reference map extrapolation creates striations in stress outside the levelset which do not get sampled until a partial grid-cell becomes a full solid cell (fig.3-8).

2. A partial cell then becomes a full solid cell and therefore now samples the stress
Figure 3-8: Stress obtained from extrapolated reference map as the solid undergoes deformation obtained from badly extrapolated reference map (fig.3-10).

3. The extrapolation of stress shoots the defect outside of the levelset therefore creating a huge stress (fig.3-9 and fig.3-11).

4. The update to velocity calculates the divergences of stress and therefore samples this huge variation.

5. The velocity field tries to compensate for the huge stress and gets a high update value causing a shock on the interface leading to a pressure wave in both solid and fluid (fig.3-12).
Figure 3-9: Solid stress with its extrapolation right before and after the interface crosses through grid edges

Figure 3-10: Zoom in on solid stress without its extrapolation right before and after the interface crosses through grid edges

### 3.5 Improving the extrapolation technique

Now that we have a clear understanding of what these blips are and how they propagate through our fields we can go to the origin and try to fix it. The extrapolation routine performs a linear extrapolation along the normal direction as defined by the gradient of the levelset as described by Aslam [1] and Rycroft and Gybou [18]. However, a question arises when calculating the slope of the values to be extrapolated. One possibility would be to project the x and y components of the gradient of the
Figure 3-11: Zoom in on solid stress with its extrapolation right before and after the interface crosses through grid edges.

Figure 3-12: Creation and propagation of a blip in the pressure field as the levelset crosses a grid edge.

Field onto the normal direction and use this as the slope to create the extrapolated values and then project back in x and y. The other approach would be to avoid the first projection and extrapolate the full gradient of the field as two separate scalar fields x and y separately and then recombine the extrapolated values, project along the normal and create the normal extrapolation.

Using the second method we obtain much better results in terms of the fact that starting from a solid in a stress free state after extrapolating the reference map we still obtain a stress free reference map in the newly created values (fig.3-13).
This definitely was a vital improvement to the implementation of the code since with the previous one we were assuming completely wrong stresses and even creating stresses where they should not exist from the get go. As you can see in fig.3-14 the difference before and after is crucial especially in the case of the rotor where the interface changes curvature a lot.

However, at this step although improved and fixed for the case of an undeformed solid there was still more to the story than that. Once the solid starts deforming the extrapolation technique still has an inherent tendency to create striations in the stress obtained from reference map extrapolation (fig.3-15).
Figure 3-14: Stress based on reference map extrapolation with the improved independent x,y extrapolation routine showing no defects in the case of the rotor simulation.

Figure 3-15: Striations in solid stress obtained from reference map extrapolation inherent to the extrapolation routine.

### 3.6 Discovering the checkerboard extrapolation error

A very interesting discovery was then made based on a very simple experiment. We defined a new field inside the solid which was composed only of checkerboard which we then extrapolated outside using the extrapolation routine and we looked at the resulting fields as the solid deforms and so does the interface curvature. We obtain the field shown in fig.3-16.
We immediately observe a strong resemblance between the resulting extrapolation and the striations which we also have in the stress fields. This lead us to question the quality of our stress fields and to wonder if they did not show checkerboarding patterns to begin with. After closely looking at the resulting fields by manually outputting them into a csv type of file format and looking at them closely value by value in excel we realized that in deed we obtained a checkerboard pattern and our plotting tool in gnuplot was actually interpolating between the exported values in order to produce images which are more pleasing to the eye. We therefore needed a way to go around gnuplot’s inherent interpolation scheme and this was eventually done by Chris Rycroft with the addition of an additional tool in our toolbox which basically converts an array we would like to plot into a new larger array where every value is written in a block of 4 points instead of only one. Thus, afterwards even if gnuplot interpolated it actually does so by sampling 4 times the same constant value.

Once this tool was put in place we discovered a whole new problem in our code. Our stress fields revealed a checkerboarding pattern as can be seen in 3-17. This was again quite confusing since the solid has an elastic constitutive behavior and should not allow such checkers to persist but even in the case where such a pattern could appear then elasticity should eventually balance it out. We were thus lead to the thought that in some way the solid was not fully “understanding” what is going
on with its fields. But where do we break the chain? Or actually, at that time we were still looking at it from the opposite perspective which is “How do we break that feedback between the fields so that these checkers cannot persist?”.

![Figure 3-17: Revealing the presence of a checkerboard pattern in the solid stress fields and their extrapolation](image)

### 3.7 Blurring techniques

The most natural approach at this point seemed that we needed to introduce a “blur” of some kind in the spirit of breaking a feedback loop. At this stage we started experimenting with convolutive methods for blurring. The simplest way to apply a blur to a field in terms of implementation is a loop which goes through every gridpoint in the field and creates a new field which at that same position contains a new value obtained by a sum of its surrounding points with some kind of weighting. Therefore, the size of the convolving array defines the radius of the blur and the distribution of the weights can define different “directions” of blurring. For instance a blurring kernel as described in 3.3 would result in a mean blur with a radius of 1 point.

\[
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9} \\
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9} \\
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9} \\
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9}
\]  

(3.3)
In the same way, we can define a Gaussian distribution inside an $n \times n$ matrix and obtain the so-called Gaussian blur. We experimented with various set-ups starting from a simple $3 \times 3$ to an $n \times n$ isotropic Gaussian blur, and to an exotic $n \times n$ anisotropic Gaussian which is much narrower in one direction than in its perpendicular one and is then oriented with its wide direction along the interface normal, thus providing a tangential blur which was supposed to remove or at least greatly reduce the tangential striations as observed in the extrapolated values. Although we were in the right path these ideas proved to be insufficient and very slow in terms of calculation time, especially in the case of the anisotropic kernel which needs to be recalculated at every gridpoint. Since we did not have a clear understanding of the origin of the checkering we were once again facing a problem which we did not know how to approach and we were thus hoping that by applying a blur step by step through every possible field we would eventually break the checkering feedback process. We were then reminded that there is no such thing as luck in numerical methods as we did not gain a great amount of knowledge through these experiments.

3.8 Modifying averaging stencils to avoid checkerboarding

We then needed to look at the problem from a different perspective and this is exactly what we did. Instead of trying to “break a loop” we realized that we were actually already breaking one too many and thus the solid did not understand that its stress is showing these funny patterns. But where did we lose that information? It is after all the divergence of solid stress which updates velocity. That was it! It is exactly the divergence of solid stress which gives us the update to velocity but how is that divergence calculated? Does it carry all of the necessary information? The answer, of course, appeared to be no. Since we have two different grids - a regular and a staggered one, where we keep different fields we sometimes face the problem of moving information from one type of field to the other. A similar problem appears in
the case of the divergence of stress. The stresses live on the staggered grid but since velocities live on the regular grid our resulting divergence of stress also ideally needs to live on a regular grid. To do so we would actually use a four corner stencil. For instance, the x derivative of the stress component is calculated as in the first image of 3-18 that is by taking a finite difference of the stresses in the top right and top left and then averaging with the finite difference of the bottom right and bottom left points resulting in a derivative which theoretically lies in the center of the gridcell. However, this averaging in the stencil actually allows for a zero energy mode in the form a checker. This is actually where we break the feedback which would inform the solid that its fields are trying to do checker and that it needs to correct them. We thus needed to use a tighter stencil and the simplest one can conceive is a triangular stencil. We chose the top right triangle as in 3-18.

![Figure 3-18: Old and new stencils for calculation of divergence of stress](image)

Similarly, the gradient of the reference map was also using a four-point stencil which we replaced with a triangular one but this time using the top left triangle as in 3-19. These two operation completely removed any checkering patterns from our fields as can be seen in 3-20.

It is true at this moment to argue that because of the asymmetry in the stencils we eventually lose some symmetry in our simulations. However, this effect is more and more reduced as the grid size gets smaller. We are also considering the possibility of
Figure 3-19: Old and new stencils for calculation of divergence of stress

Figure 3-20: Comparison of the pressure field when no solid viscosity is applied before and after the stencil modifications

modifying our grid structure towards a setup more similar to a MAC grid in order to avoid checkering but preserve the symmetry and allow for second order convergence of the scheme (see section 4.10 for a further discussion on convergence).
Chapter 4

The blurred interface implementation for fluid-solid coupling

4.1 Preamble

This chapter focuses on our new method for implementing the reference map technique for fluid-solid interaction and is written as an independent entity meant to be read separately from the rest of this thesis.

4.2 Introduction

In this work we will present a blurred interface method for fluid-solid interaction (FSI) based on the reference map technique as presented by Kamrin and Rycroft[12]. We will present its capabilities of doing fully coupled simulations of a compressible Navier-Stokes fluid and highly non-linear solid undergoing large deformations all performed on a single Eulerian grid.

Solving a fully coupled problem of fluid-solid interaction has long interested people. However, from a constitutive point of view it is easier to solve a fluid problem on an Eulerian grid[5],[22],[23],[19],[9], since its stress is obtainable from the
finite-difference of a velocity field on a fixed mesh and a solid in a Lagrangian framework\cite{25},\cite{21},\cite{10},\cite{2} where stress depends on the total deformation and is computable from the relative positions of neighboring nodes. People have thus come up with various approaches to solving the coupled problem. Up to date, work has been done in Lagrangian-Lagrangian methods \cite{6},\cite{24},\cite{17}, that is both the fluid and solid are solved in a Lagrangian framework which potentially causes problems with meshing and remeshing as the fluid starts to flow. Other approaches avoid this problem by adopting an Eulerian-Lagrangian \cite{6},\cite{24} scheme which solves the fluid phase on a fixed grid and the solid using a Lagrangian approach such as FEM. The problem with this idea is the necessity to perform complicated projections between phases on different grids exactly on the interface with is from the beginning the most delicate piece of any FSI code. Some advantages of a fully Eulerian method can thus be directly seen. An explicit Eulerian method would provide a more efficient code in terms of calculation time, everything would be solved on the same fixed mesh so there would be no trouble in dealing with meshing and remeshing. Topological changes would be easier to take into consideration as much work has already been done on tracking interfaces with levelsets \cite{19},\cite{16} which work very well with any topology. Furthermore, multiscale and multiphysics coupling would also have advantages when done an Eulerian grid. One specific example which will be later discussed in more detail is the case of multiple solids immersed in a fluid, which we sometimes refer to as FSSI for Fluid-Solid-Solid Interaction. Finding contact between two solids on a Eulerian grid is almost as easy as asking a gridcell whether it believes to be in solid one and if so whether it also believes to be in solid two, whereas, finding contact or even distance between two meshes of a finite element kind of approach would roughly require a least squares type of projection between every node of one solid and every edge of the other one.

The Reference Map Technique (RMT) is such a fully Eulerian framework which has already been implemented in a sharp interface approach \cite{12},\cite{13}. The work done by Kamrin et al. clearly demonstrated the capability of solving non-linear finite strain problems in a solid using a fixed mesh, as well as a first attempt to
solving a fully coupled problem of FSI. However, the blurred implementation which we present here is a simpler implementation, using fewer numerical extrapolations and therefore introducing less of the numerical artifacts inherent to the extrapolation routine. From a physical point of view, sharp methods have their advantages - they do show a distinct separation between each phase regardless of grid size, whereas a blurred interface method has a transition zone between the properties of each phase which needs to be considered as the interface altogether. Even if at first glance, the blurred interface method would seem as a compromised approach it actually has significant advantages which we will present in this work.

The sharp interface technique is based on work done by Fedkiw [7] on fluid-fluid interactions based on the Ghost Fluid Method (GFM). This was then used a basis for a sharp technique for fluid-solid interactions. However, in order to satisfy jump conditions (such as continuity of normal velocity, normal and shear tractions) across a boundary we need to create trustworthy extrapolations of all kinematic and all stress fields from each phase to the others (i.e. solid velocity is extrapolated inside the fluid to create the so called solid ghost velocity values and fluid velocity is extrapolated into the solid as to create the fluid ghost velocity). The keyword here being trustworthy. The validity of a sharp interface method is basically limited by the quality of the extrapolation technique and its ability to create a truly predictive extrapolation. Over time this becomes a progressive problem as you keep trusting many extrapolations repeatedly and with new values getting created at every time step. Although in fluid-fluid codes this does not show up as a considerable problem due to their inherent dissipation, in the case of a fluid-solid interaction code where the solid does not have viscosity it immediately poses serious problems revealing themselves as small glitches or “blips” on the interface whenever the interface crosses through grid cell boundaries. In a sharp technique extrapolation error can be accumulated, particularly in the case when the object acquires new grid cells in its domain requiring for numerical tricks such as artificial damping (viscosity) or surface tension which penalizes the kind of error introduced by the extrapolations.

A blurred interface method on the other hand, utilizes the best from both worlds.
A fluid-fluid blurred technique [3],[15] does not require any extrapolation. In the case of a fluid-solid problem, using only one extrapolation - that of the reference map (described in more detail in section 4.3.1), we are able to solve a finite strain solid on a fixed grid by having a single velocity field in the whole computational domain. This avoids the necessity for multiple extrapolations, therefore, creating a potentially faster numerical scheme. In addition, jump conditions do not need to be explicitly applied but are implicitly met inside the *transition zone*.

### 4.3 Concept

The fundamental difference with the sharp interface method is that, as the name suggests, there is no more sharp distinction between the fluid and solide phase(s). Instead, there is a region in between the solid and the fluid where the properties gradually change between phases. This region is what we refer to as the *blur zone* or the *transition zone*. The width of the blur zone (which is a function of the grid size) then becomes one of the parameters of the simulation and in the results presented later on we will plot the phases with a boundary representing the thickness of the blur zone. The whole blur zone should be considered as the actual interface. It is true that we lose some information which we used to have with the sharp distinction, however we also gain a lot in terms of simplicity and rapidity of calculation. In addition, a faster calculation would allow us to reduce the grid size which effectively gets us closer to a sharp interface.

#### 4.3.1 The Reference Map

People familiar with finite element methods (FEM) will recognize its similarity to the motion function which maps a point from the reference (undeformed) configuration to its position in the deformed one as illustrated in figure 4-1.

The reference map is actually the inverse concept - we map the points from the deformed configuration into the reference one as in 4-2.

As described by Kamrin et al. [12] in an Eulerian frame the conservation of mass
and momentum in strong form (when deformations remain smooth) can be written respectively as in 4.1 and 4.2.

\[
\begin{align*}
\rho_t &= -\nabla \dot{v} - \rho \nabla \cdot \dot{v} \\
\dot{v} &= -\nabla \cdot \dot{v} - (\nabla \cdot \mathbf{T} + pg)/\rho
\end{align*}
\] (4.1) (4.2)

Where \( \dot{v}(x,t) \) is the spatial velocity and \( \mathbf{T}(x,t) \) is the Cauchy stress tensor field. As previously discussed, we can express the motion of an element of material initially at \( \mathbf{X} \)

---

[Figure 4-1: Motion function]

[Figure 4-2: Reference map function]
moving to \( x \) at time \( t \) as \( x = \chi(X, t) \), therefore, defining the deformed configuration \( B_t \). The motion then defines the deformation gradient \( F \),

\[
F(X, t) = \frac{\partial \chi(X, t)}{\partial X}
\]  

(4.3)

In our case, since the method is Eulerian we cannot use the motion function but instead use the reference map which we define as

\[
X = \xi(x, t)
\]

(4.4)

Using the chain rule it can then be shown that the deformation gradient \( F \) is obtainable as in 4.5.

\[
F(X, t) = (\nabla \xi(x, t))^{-1}
\]

(4.5)

Once we have the deformation gradient we are capable of modeling any thermodynamically compatible solid law on a fixed grid. Throughout this work we have consistently used hyperelastic stress as simply

\[
T = 2(\text{det} F)^{-1} F \frac{\partial \hat{\psi}(C)}{\partial C} F^T |_{F = (\nabla \xi)^{-1}, C = (\nabla \xi)^{-T}(\nabla \xi)^{-1}}
\]

(4.6)

Here, \( \psi \) is the Helmholtz free-energy per unit (undeformed) volume and \( ^\hat{\cdot} \) designates a function, therefore

\[
\psi = \hat{\psi}(F)
\]

(4.7)

The specific model applied in this work has always been the compressible neo-Hookean elastic solid model. The small-strain bulk and shear moduli are taken to be \( \kappa = 10 \) and \( G = 10 \) respectively. Letting \( J = \text{det} F \) and indicating the deviator with \( ' \), we define \( \psi \) as in 4.8 and the Cauchy stress tensor is then given by equation 4.9.

\[
\psi = \frac{G}{2} (J^{-\frac{2}{3}} \text{tr}(B) - 3) + \frac{\kappa}{2} (J - 1)^2
\]

(4.8)

\[
T = G J^{-5/3} B' + \kappa (J - 1) \mathbf{1}
\]

(4.9)
For more information on general finite-strain elasticity refer and hyperelasticity refer to [8]. We now also need an Eulerian update rule for the reference map $\xi$. We remind that the reference map tells us the origin of the current particle position, thus, we can observe that the reference map for a tracer particle never changes, giving us an advective evolution law

$$\xi_t + \mathbf{v} \cdot \nabla \xi = 0$$  \hspace{1cm} (4.10)

or in conservative form,

$$(\rho \xi)_t + \nabla \cdot (\rho \xi \otimes \mathbf{v}) = 0.$$  \hspace{1cm} (4.11)

In the case where the initial configuration is undeformed the initial condition for the reference map is just

$$\xi(\mathbf{x}, t = 0) = \mathbf{x} = \mathbf{X}.$$  \hspace{1cm} (4.12)

The current implementation of the method is such that we utilize a regular, structured grid where we keep the kinematic quantities $\xi$ and $\mathbf{v}$ and a staggered grid with nodes in the cell centers of the regular grid where we keep the resulting fields at each time step - the deformation gradient $\mathbf{F}$ and the Cauchy stress tensor $\mathbf{T}$ (see fig. 4-3).

In fig.4-4 we present the outline of the method allowing us to move forward in time, starting from a time step $n$ with given primary fields $\mathbf{v}^n$ and $\xi^n$, going to time step $n + 1$ and calculating $\mathbf{v}^{n+1}$ and $\xi^{n+1}$.

![Figure 4-3: Storage of quantities on a regular and staggered grid](image-url)
The "basic" RMT routine; $Eu(\xi^n, v^n)$, strong formulation.
(All $\nabla$ operators discretize appropriately to centered or WENO finite-differences.)

Given: $v^n$ and $\xi^n$

Goal: Calculate $v^{n+1}$ and $\xi^{n+1}$

Step 1: Construct $F$

Step 2: Compute $\rho$

Step 3: Compute $T$

Step 4: Update $v$

Step 5: Update $\xi$

$F = (\nabla \xi^n)^{-1}$

$\rho = \rho_0 (\det F)^{-1}$

$T = T(F)$

$\frac{v^{n+1} - v^n}{\Delta t} + v^n \cdot \nabla v^n = \frac{1}{\rho} \nabla \cdot T + g$

$\frac{\xi^{n+1} - \xi^n}{\Delta t} + v^n \cdot \nabla \xi^n = 0$

---

Figure 4-4: Outline of the Reference map technique

### 4.3.2 The computational domain

We now have all of the ingredients to define our computational domain. Looking at fig.4-5 we start by constructing a regular grid where we keep our primary fields - the kinematic fields $v$ and $\xi$, a staggered grid where we keep the resulting deformation gradient and stresses at every time step, and a function $\phi$ whose zero level-set describes the interface (the boundary) between phases. The $\phi$ function (sometimes just referred to as the levelset) is also kept onto the regular grid. It has the useful feature to not only represent the boundary locus but also represent a distance measure between any given point and the interface. Since $\phi$ lives on the regular grid it is therefore very easy to distinguish between regular grid points which are inside the solid and those which are inside the fluid by just looking at the sign of $\phi$ as described in section 4.3.4. It is important to point out that when considering the staggered grid or in other words the cells formed by the points of the regular grid we have some which get crossed through by the interface. Whereas, in the sharp interface method we used to have trouble defining the nature of these cells and choosing which constitutive model to apply in the case of the blurred interface everything is done much simpler as described again in section 4.3.4.
We construct:
- Regular grid
- Staggered grid (+)
- Level Set

We can define:
- A solid region
- A fluid region
- A partial solid/fluid region

Figure 4-5: The computational domain of our simulations: regular grid (white), staggered grid (red crosses), interface (white curve) defining a solid region (green), a fluid region (blue) and a partial solid/fluid region (orange)

4.3.3 The new stencils for rmap gradient and stress divergence

A regular grid value is referenced as a $[ij]$ position and goes from 0 to $(n-1) \times (m-1)$ for a total of $n \times m$ values, whereas a value from the staggered grid is referenced as $[ije]$ and ranges from 0 to $(ne-1) \times (me-1)$ for a total of $ne \times me$ values. The $[ije]$ node is the staggered grid-point in the center of the current cell. The $[ij]$ node is the top-right-hand corner of that cell and therefore, the $-m$ term gives us the lower neighbor and thus making $[ij - m - 1]$ the lower-left-hand corner of that cell.

In order to avoid problems with checker-board instabilities inside the solid we use a very tight stencil in order to calculate the divergence of stress terms and the gradient of rmap as in fig.4-6 and fig.4-7. The actual discretized version of the gradient of rmap is therefore as given in eq.4.13 and eq.4.14. And similarly the definition of divergence of stress $\nabla \cdot \mathbf{T}$ is given for the $\sigma_{11}$ and $\sigma_{12}$ components in eq.4.15 and eq.4.16. The factor allowing us take the neighboring value from the point below in this case is $-me$ since the staggered grid has one more point per direction than the regular one.

$$\frac{\partial \xi_x}{\partial x} \bigg|_{[ije]} = \frac{\xi_{[ij-m]} - \xi_{[ij-m-1]}}{dx} \quad (4.13)$$
\[
\frac{\partial \xi_x}{\partial y}_{[ij]} = \frac{\xi_{[ij-1]} - \xi_{[ij-m-1]}}{dy}
\] (4.14)

\[
\frac{\partial \sigma_{11}}{\partial x}_{[ij]} = \frac{\sigma_{11}[ij + me + 1] - \sigma_{11}[ij + m]}{dx}
\] (4.15)

\[
\frac{\partial \sigma_{12}}{\partial y}_{[ij]} = \frac{\sigma_{12}[ij + me + 1] - \sigma_{12}[ij + m]}{dy}
\] (4.16)

Figure 4-6: Old and new stencils for calculation of divergence of stress

Figure 4-7: Old and new stencils for calculation of the gradient of reference map
4.3.4 Mixing quantities across the interface

We start by defining a function $\phi$ whose zero levelset represents our solid phase as can be seen in 4-8. This function defines not only where the locus of the interface as per 4.17 but also represents a measure of distance from the interface since the method [4] which moves the levelset by construction always keeps its slope to 1.

$$\begin{cases} 
\phi < 0, & \text{the point is inside the solid phase} \\
\phi > 0, & \text{the point is outside the solid phase.}
\end{cases}$$ (4.17)

Figure 4-8: The zero contour of the levelset as well the $\phi$ function

Next, we need to construct the adapted error function, or from now on referred to as simply the error function. The error function is the rule defining how one phase crossfades into the others. The choice of using the error function was due to the simplicity of obtaining a smooth continuous function, going from 0 to 1 symmetrically around the interface and have horizontal tangents a few gridspacings away from the interface. The mathematical definition of the error function is as defined in 4.18. We then use a shifted and normalized version of it to give us the required function as in 4.19. Where $\frac{\phi(x)}{n_b \times dx}$ is a measure of distance which goes to 1 at $n_b$ gridsizes away from the interface, therefore, defining the distance at which the adapted error function reaches roughly 84 percent.
Figure 4-9: The definition of the error function, the adapted error function and the adapted error function around the interface in a 2D problem

\[ erf(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt \]  \hspace{1cm} (4.18)

\[ erf'(x) = \frac{1}{2}(1 + erf(\frac{\phi(x)}{n_b \times dx})) \]  \hspace{1cm} (4.19)

We now have a straightforward way of defining the mixture of quantities everywhere in our computational domain based on the error function field \((erff)\). For instance, the \(\sigma_{11}\) stress component becomes a mixture of the stress obtained from the fluid constitutive law applied to the whole domain and the stress obtained from extrapolated reference map (denoted by the superscript xe as in \(\xi\) extrapolation), weighted respectively by \(erf\) and \(1 - erf\) (see 4.20).

\[ \sigma_{11}[ije] = erff[ije] \times \sigma_{11}^{f}[ije] + (1 - erff[ije]) \times \sigma_{11}^{xe}[ije] \]  \hspace{1cm} (4.20)

### 4.3.5 Outline of the method

We can now describe how the blurred method works in order to step forward in time. Starting at a time step \(n\) in order to obtain the kinematic fields for the next step \(n + 1\). The resulting stress fields are therefore recalculated at every time step.
Data: \(v^n\) and \(\xi^n\)

Result: \(v^{n+1}\) and \(\xi^{n+1}\)

1. Calculate the update to reference map (eq.4.21);
2. Calculate the fraction of each phase in the whole domain (sec.4.3.4);
3. Compute solid stress wherever we have \(\xi\) (sec.3.8);
4. Compute the non-viscous component of fluid pressure due to density change. This is applied in the whole domain (eq.4.23);
5. Add fluid viscous stress terms and apply mixing rule to stress and density (eq. 4.24, sec.4.3.4, sec.4.7 respectively);
6. Calculate the update to the velocity fields and fluid density (eq.4.28);
7. Apply update to reference map, velocity and fluid density;
8. Set boundary conditions (such as anchoring);
9. Extrapolate reference map for the next time step and apply diffusion to extrapolated values (sec.4.8);

**Algorithm 1**: Outline of the blurred interface method for FSI

The update to reference map is based on the discretized version of eq.4.10 and is given in eq.4.21. The gradient of \(\xi\) term \(\nabla \xi\) is calculated using ENO [20].

\[
\xi_{[ij]}^{n+1} = \xi_{[ij]}^{n} + dt(-v_{[ij]}^{n} \cdot (\nabla \xi)_{[ij]}^{n})
\]  
(4.21)

The update to velocity is as given by 4.22.

\[
v_{[ij]}^{n+1} = v_{[ij]}^{n} + dt(v_{[ij]}^{n} \cdot \nabla v_{[ij]}^{n} + \frac{1}{\rho_{[ij]}^{m}} \nabla \cdot T_{[ij]}^{n} + g)
\]  
(4.22)

Where the divergence of stress is discretized as in eq.4.15 and eq.4.16. And the velocity gradient also uses ENO [20].

The non-viscous fluid stress is obtained by 4.23 where \(\lambda_f\) is the fluid compressibility modulus, \(\rho_0^f\) is the initial density and \(\eta\) is fluid viscosity.

\[
p_{[ij]}^{n} = \lambda_f \left( \frac{\rho_{[ij]}^{n} - m - 1}{\rho_0^f} - 1 \right)
\]  
(4.23)
The full fluid stress tensor satisfies eq. 4.24.

\[ T_f = \eta \frac{\nabla v_f + (\nabla v_f)^T}{2} - \lambda_f \rho_f \left( \frac{\rho[ij - m - 1]}{\rho_0} - 1 \right) \mathbf{1} \]  

(4.24)

The discretization of the gradient of velocity \( \nabla v \) is done similarly to the reference map gradient described in sec. 3.8 or to be more precise as defined by eq. 4.25 and eq. 4.26.

\[ u_x[ij] = \frac{u[ij - m] - u[ij - m - 1]}{dx} \]  

(4.25)

\[ u_y[ij] = \frac{u[ij - 1] - u[ij - m - 1]}{dy} \]  

(4.26)

4.3.6 Reduced number of extrapolation steps

As previously mentioned the difference between the blurred and the sharp interface method goes beyond how we define the change between phases but there is an actually a fundamental difference. The blurred method is based on the idea that we use a unique velocity field for the whole computational domain. Therefore we do not need to create ghost values anywhere. While there is still a step of extrapolation for the reference map values, the approach is rather different than the GFM [7] since we do not extrapolate from one phase into the others and the other way around thus requiring to validate jump conditions on the interface. The logic of the blurred interface method is then that because we have velocities everywhere we can actually apply the fluid constitutive law in the full domain (even inside the solid phase(s)) and we can apply the solid constitutive law everywhere where we have values for the reference map - that is inside the solid and inside the narrow band surrounding it. This change in the concept significantly reduces the number of extrapolation steps required for the method to work. Whereas, in the sharp approach (in a 2d case) we needed to extrapolate 3 solid stress components, 3 fluid stress components, 2 solid velocities components, 2 fluid ones, fluid density, and the 2 components of the reference map, resulting in a total of 13 scalar fields which need to be extrapolated for each solid phase, in the blurred interface method we actually only need to extrapolate the 2
reference map components for every solid phase. This greatly decreases the number of calculations done per timestep and improves efficiency, especially, when we consider problems of higher complexity involving multiple solids immersed in a fluid. In that last case, the blurred interface method has yet another advantage which is the ease of not having to explicitly set jump conditions.

4.4 The importance of the reference map extrapolation

Our first implementation of the blurred interface method inside the framework of our previous sharp method was very instructive to confirm our understanding of the importance of the reference map extrapolation. While solving “blip” issues in the sharp method we gradually got to the point to actually outputting what solid stress fields result outside of the solid boundary based on the reference map extrapolation. These values are extremely important for an Eulerian method such as the Reference Map Technique because, when one thinks about it, the solid never actually moves but we change the identity of the adjacent grid cells when we require the solid to actually "move". This means that the stress obtained from extrapolated reference map is actually our guess for what the stresses would be if the solid was to move to a particular region. Therefore, as the solid acquires new grid cells inside its boundary the stress obtained from extrapolated reference map at this location actually just becomes the stress obtained from reference map or what we usually call our full solid stress.

Even though, we had already greatly improved our extrapolation routine and it did not anymore create stresses in the case of a stress free state of the solid, we still have a problem with the striations which in the resulting fields as the interface changes curvature. We have previously followed the complete logical chain and method of propagation of how these striations create “blips” in the sharp interface method. Another way to consider the whole problem in that case was that we begin with
the solid residing in a certain region thus occupying a certain set of gridcells. We then extrapolate values for the reference map outside this set but do not ever sample these values until the moment when a new gridcell is acquired by the solid. At this moment, the actual solid stress obtains a value which up to now was a stress obtained from extrapolated reference map thus feeling a high jump in stresses due to the extrapolation striations. In a way we could say that the method was making up wrong values but was ignoring them until the moment when it had no other choice but to use them and thus creating a sudden “blip”. The blurred interface, however, would gradually see these new values approach and the dynamics of the method should equilibrate for them over time. We can see the resulting solid stress field in 4-10. We can actually see that without our previous numerical “band-aids” such as surface tension and solid viscosity the interface actually starts to wrinkle gradually and you can fully understand what the underlying reason for that is when you look at the stress obtained from extrapolated reference map in 4-11. All of the wrinkles actually correspond to striations in the extrapolation and vice versa. The question of which one is first and which one is causing the other is similar to the chicken and egg question. At this time it is also important to point out that the extrapolated reference map values are recreated at every timestep and thus do not actually benefit from the dynamics of the elastic solid constitutive law to balance striations out. This is the actual reason why wrinkles persist over time and we the sharp method would always feel shocks when sampling outside values.

4.5 Discrepancies between the interface locus as per RMAP and as per levelset

We where then again drawn back to the question why the solid does allow its interface to adopt such wrinkles when in fact it is an bulk of elastic material which should minimize its surface energy. We then realized that we could actually “ask” what the solid thinks about this by looking at the interface as defined by the reference map
Figure 4-10: The solid 11 stress as obtained with the first implementation of the blurred interface method instead of as it is defined by the levelset. We remind that the reference map tells us where we think material at a current point originated from. This means that if we apply the current reference map to the initial definition of the levelset we can actually understand where the current interface is according to the solid. Making a reference to the levelset being called $\phi$ we decided to call the interface as per reference map $\psi$ where it is defined as in 4.27.

$$\psi(x, t) = \phi(\xi(x, t), t = 0) \quad (4.27)$$

Even if the two fields reference map and levelset were supposedly updated in a very similar fashion - through advection based on the same velocity field we realized that
there were serious discrepancies in where the two fields believe is the actual interface. Looking at the comparison between them in 4-12 we realize that error builds up over time in the locations where the reference map extrapolation goes bad and that we need a mechanism to keep them together at all times as to inform the levelset that it is actually surrounding a solid with a certain constitutive law and not just a region of material which gets advected by a velocity field. One could argue at this moment that we could actually always reference everything in terms of $\psi$ but the levelset has the great advantage that it not only defines the interface with its zero levelset locus but also gives a measure of distance as its slope is always $1$. 

Figure 4-11: The stress obtained from extrapolated reference map showing correlation with the wrinkles forming on the interface
Figure 4-12: Drift between the interface as described by the levelset (white contour) and as described by the reference map (blue contour)

### 4.6 Improved pinning technique

The improved pinning technique is a tool which makes sure that the levelset always gets "pinned" onto the solid. In other words, $\psi$ does not get advected by velocity anymore, as it is just a referencing tool and no mechanics is actually connected to its movement, but it gets advected onto $\phi$ thus always assuring that afterwards whenever we ask the levelset $\phi$ if a specific point is inside or outside the solid there is no risk of confusion. In our previous setup since the levelset was trying to wrinkle in places where the reference map remains smooth these actually meant that we had “pockets” of material which are inside the solid according to $\psi$ but inside fluid according to $\phi$ and since $\phi$ is used when choosing which constitutive law to apply meant that the
wrong one was applied to those regions where the two fields do not agree.

The idea behind the new pinning technique is pretty straightforward. Since we want to always advect $\phi$ onto $\psi$ we need a velocity field which gives us the speed in the normal direction at which to move $\phi$. One such measure is basically just the difference between the two fields $\phi - \psi$. As the two get closer so does the difference between the two scalar fields and therefore so does the correction and eventually $\phi$ converges towards $\psi$. The great thing behind this idea is that we already have a good way of moving $\phi$. Using the Chopp [4] method for moving the $\phi$ levelset always makes sure that its slope is kept to 1. Or in other words $\phi$ is always a solution to the Eikonal equation.

**Data:** The position of the interface according to the level-set $\phi$ and according to the rmap $\psi$

**Result:** $\phi$ is moved to match its zero contour with the one of $\psi$

while criterion based on the difference is not met do
  Calculate the difference $\phi - \psi$;
  Update level-set using fast-marching[4] with a normal speed given by
  $u_n = k_{pining}(\phi - \psi)$;
end

**Algorithm 2:** Improved pinning technique

### 4.7 Density mixing and density update

An important concept with the blurred interface method as compared to the sharp one is that since we have a unique velocity field everywhere, instead of calculating fluid density in the full fluid and extrapolate it into the solid to meet boundary conditions we can actually calculate a fluid-like density everywhere and update it every time step following equation 4.28, where the gradient of density $\nabla \rho$ is obtained through ENO [20] and $\nabla \cdot \mathbf{v}$ is obtained by centered differences. This allows us to avoid the need of doing an extra step of extrapolation and therefore reduces the amount of numerical error which we accumulate.
\[ \rho_{ij}^{t+dt} = \rho_{ij}^t + dt(-v_{ij} \cdot (\nabla \rho)_{ij}^t) - \rho_{ij}^t \nabla \cdot v ) \quad (4.28) \]

The solid density on the other hand is always recalculated at a new time step by using the determinant of the deformation gradient which is stored everywhere inside the solid and inside the narrow band.

We now require once again a rule for mixing these two quantities and defining what the global unique density field is. In order to do so we apply a similar approach to the one for stress as defined in 4.20. We therefore define the mixed density \( \rho_m \) as in 4.29.

\[
\rho_m[ij] = erf_{ff}[ij] \times \rho_{ij}^f + (1 - erf_{ff}[ije]) \times \frac{1}{4}(\rho_s^{\sigma c}[ije] + \rho_s^{\sigma c}[ije+1] + \rho_s^{\sigma c}[ije+me] + \rho_s^{\sigma c}[ije+me+1]) \quad (4.29)
\]

The density is a field which is stored on our regular grid for the fluid and on the staggered one for the solid thus explaining the difference in indices. A regular grid value is referenced as a \([ij]\) position and goes from 0 to \((n-1) \times (m-1)\) for a total of \(n \times m\) values, whereas a value from the staggered grid is referenced as \([ije]\) and ranges from 0 to \((ne-1) \times (me-1)\) for a total of \(ne \times me\) values. You can therefore notice that in order for the mixture of fields to be created in the correct position and not translated by half a grid-size, as the staggered grid points are in the centers of cells formed by the regular grid, then we need both a regular version of the error function field \(erf_{ff}\) and a staggered one \(erf_{sf}\). Should the subscript be omitted you can know which version is used by checking whether it is an \([ije]\) or an \([ij]\) reference.

### 4.8 Applying diffusion to the extrapolated values of the reference map

Once we realized how important the extrapolation of the reference map is for not only this code but the general concept of a solid solved in an Eulerian framework, we needed to create a smoother extrapolation method. Given that our extrapolation
only creates new values along the interface normal direction it never actually samples its neighboring values and therefore is left completely unconstrained in the tangential direction. This results in what we have already observed and defined as “striations” in the extrapolated values. Furthermore, by applying the checkerboard extrapolation test case as in 3-16 we realized that our approach does not require a perfect extrapolation defined as a mathematical tool. In fact, we need a smooth extrapolation - one which is good initial guess of what stresses would be there if the solid were to acquire new points and one which would not induce any shocks whenever new grid cells are acquired. In the case of the blurred interface method this approach is even more logical as the extrapolation is indeed merely an initial guess which then gets refined as the new values gradually start to take part of the mixed values inside the transition zone.

**Data:** A given solid field \( f \) and its extrapolation

**Result:** The given field and its smooth extrapolation

Calculate the laplacian of the field and find the largest curvature

while criterion based on the largest curvature is not met do
  | Evaluate the laplacian of the field
  | Create new temporary field \( X_t[ij] = f[ij] + k_{diffusion} \nabla^2 f \)
  | Replace the values of \( f[ij] \) with \( X_t[ij] \) for every \( [ij] \) outside of the solid
end

**Algorithm 3:** Diffusion step on the extrapolated values

As can be seen in the test case of the checker which gets extrapolated, the extrapolated values are diffused in pseudo-time to smooth out any numerical striations and give us a more reasonable guess for what the extrapolated field should be.

Let us now look at an actual application in a simulation. In figures 4-15 and 4-16 we see a comparison between the \( \sigma^{ref}_{11} \) stress component without diffusion and with diffusion. The first one exhibits striations in the extrapolated values whereas the second one remains smooth all throughout the simulation. Furthermore, in the first case the interface is still trying to wrinkle and this causes a ripple in stress as can...
Figure 4-13: Verification of the diffusion algorithm making sure that we obtain a smooth extrapolation of a field

be seen in 4-17, whereas with the routine applied to the extrapolated values stresses remain smooth as in 4-18.

4.9 Results

With all of these ingredients in place we now have a fully functional fluid-solid interaction (FSI) routine which is capable of doing finite strain deformations in any type of thermodynamically compatible non-linear solid fully coupled with a compressible Navier-Stokes type of fluid. We are capable of running the simulations which would previously exhibit “blips” and flickering around the interface even in the presence of
numerical "band-aids". We have removed all surface tension from the solid phase, as well as the artificial viscosity which was needed to make the simulations stable. We have also reduced the number of extrapolation steps significantly, we do not need a specific routine to ensure boundary conditions and we have introduced a new method of moving the level-set $\phi$ so that it is always in coherence with the actual underlying solid as represented by the interface $\psi$ as defined per the reference map thus ensuring that the interface is perfectly aware of the constitutive behavior of the solid it is representing.

With this improved implementation we can now model a highly deformable stick of a rubbery material which is anchored at one end and gets deformed up to very large...
deformations because of a fluid inflow condition on the left edge of the computational domain and an outflow on the right one. The same case when using the sharp interface technique required both surface tension and solid viscosity to run and even so it would randomly get pinches of the interface ("blips") and pressure waves emitting from them. The new simulation runs smoothly and does present any of the previous numerical artifacts.

Similarly, another test case which was very difficult to simulate with the previous implementation was the rubbery rotor anchored around a pivot at its center which gets deformed by a fluid inflow in the bottom half of the left edge of the computational domain and an outflow in the top right half edge of the domain. This simulation would
Figure 4-16: Stress field $\sigma_{11}^{re}$ from extrapolated reference map with diffusion resulting in a smooth extrapolation

show very strong flickering around the interface as the particularity of the shape is that curvatures varies a lot therefore creating a very complicated shape to extrapolate from. As can be seen from figures 4-20 and 4-21 no such problems are apparent even without all of our previous numerical tricks. For all fairness, it is important to point out that we do rely on a diffusion step but it only affects extrapolated values which were already a guess to begin with. The proper way to think about the diffusion step of the extrapolated values is in fact a way to give the extrapolation technique the extra information that it needs to stay smooth and thus give us a better educated guess.
Figure 4-17: Stress field $\sigma_{11}^{xx}$ from extrapolated reference map without diffusion showing a sudden ripple in stress due to a jump in extrapolated values

### 4.10 Convergence

In order to analyze convergence one would usually take a well-known problem where an analytical solution exists and compare the numerically obtained values against it. However, such a simple problem in fluid-solid interactions does not exist and we therefore have to look at the convergence of our simulations as we reduce the grid-size. We have carried out simulations on multiple grids: 60x60, 72x72, 90x90, 120x120, 180x180, and 360x360\(^1\). We chose the rotor simulation as a test case and we took the

\(^{1}\text{Remark: The predominant CFL criterion in these simulations proved to be the fluid viscosity CFL which is proportional to } dx^2 \rho \sqrt{\tau} / 2\eta \text{ so the time step is scaled as } \sim dx^2. \text{ The other criteria which need to be considered are the one due to compressibility of the solid } \sim dx \sqrt{\rho \sigma / \kappa}, \text{ the one due to compressibility of the fluid } \sim dx \sqrt{\rho \sigma / \lambda}, \text{ and the convective CFL } \sim dx / <u>.
Figure 4-18: Stress field $\sigma_{11}^{ref}$ from extrapolated reference map with diffusion not showing any ripples

Simulation all the way to $t = 4s$ in order to make sure that there has been significant interaction between the fluid and the solid and that the solid is highly deformed. A comparison of the resulting pressure fields for every respective grid can be seen in fig.4-24 and fig.4-25. We then compare every simulation against the most refined one (which we treat as an “exact” solution) and we take the log of the discrete $L^2$ norm of the difference (eq.4.30) which we plot against log(dx). The discrete $L^2$ norm of the difference between a numerical function $f_{\text{numerical}}$ and its exact counterpart $f_{\text{exact}}$ is
Figure 4-19: Resulting pressure field simulation of a stick anchored at one end getting deformed to large strain by the surrounding fluid flow given in eq.4.30.

\[
\|f_{\text{numerical}} - f_{\text{exact}}\|^2 = \sum_{ij=1}^{n \times m} (f_{\text{numerical}}[ij] - f_{\text{exact}}(x(ij), y(ij)))^2 \, dx \, dy \quad (4.30)
\]

In this way we obtain fig.4-23. We can see, first of all, that the simulations do indeed converge thus verifying that this is a successful numerical method. The next important detail is that the slope (the rate of convergence) is a bit higher than first order but is not second order. It is interesting to point out at this moment, that the Reference map technique as presented by Kamrin and Rycroft [12] showed second order convergence, however in this implementation we have changed stencils.
Figure 4-20: Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid

for divergence of stress and gradient of rmap. Since the new stencils are triangular as presented in section 3.8 they are slightly off centered thus reducing accuracy a bit as a trade-off for stability. In the future new stencils as well as a new grid setting will be examined in order to allow for the full potential of a second-order converging FSI scheme based on the Reference map technique. It is also interesting to zoom into a particular region of the simulation and compare how it looks like with a very coarse grid and with a more refined one. In fig.4-26 we can see a comparison of the upper right concave part of the rotor with the 60x60 grid and the 360x360 grid. We have also examined the convergence in the other primary field of our simulation
Figure 4-21: Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid which is the reference map $\xi$. In order to do so, we have carried out the exact same simulation as for the velocity, however since the reference map does not live on the entire domain in order to be able to compare we chose a particular rectangular region in all simulations and calculated the $L^2$ norm of the error. We thus obtained fig.4-27. We can thus see that the rate of convergence is $\approx 1.5$ in this case. It is again lower than the rate of 2 which was cited by Kamrin and Rycroft[12] just for the solid part of the Reference map technique, however, similarly to the velocity convergence this is due to the fact that we have off-centered derivatives which reduce accuracy but gain stability.
Figure 4-22: Resulting pressure field simulation of a rotor anchored onto a pivot at its center getting deformed to large strain by the surrounding fluid flow and then getting spun by the fluid.

Using our previous method we are also capable of analyzing the convergence of the stresses by taking finite differences of our previously obtained sections of reference map as per eq. 4.13. We thus obtain the convergence plot in fig. 4-28. We can see that similarly to what was discussed in [12], the convergence of the stresses is close to the convergence of the reference map. This is a very pleasing result, as it shows that both our kinematic and our stress fields have at least first order convergence, proving that our method is indeed capable of solving a highly complex problem such as fully-coupled fluid solid interactions. Furthermore, giving the actual simplicity of the code and the nice property of it being Eulerian and explicit it is fair to argue
that the method is very promising and given its rapidity large scale problems would not be a problem. Simply looking fig. 4-26 it is remarkable how well the simulations copes with a curved interface given the actual under-refinement of the mesh (the box underneath the figure is the actual grid-cell size).

4.11 Simulation of contact between two solids immersed in a fluid

With all of this framework already in place and running smoothly, going the extra step and inserting a second solid phase as to perform a fully coupled fluid-solid-solid-interaction was a matter of simply defining a second set of fields for the second level-set, solid stress fields, reference map. The velocity field is still one single unique field
Figure 4-24: Pressure field at frame $t=40$ for the rotor simulation with a grid of 60x60, 72x72, 90x90 and 120x120 grid-points.

for the whole computational domain. The only actual additional reflexion required was to define the new mixing rule which would now need to be able to cross-fade between the two solids and fluid, as well as eventually adding an extra routine to correctly set contact conditions and avoid any cross-penetration of the solid phases.

4.11.1 Formulation

The simplicity of the blurred interface technique lies in the concept that whenever possible we avoid defining any “if-cases”. That is, we aim to treat the whole domain as one - instead of multiple velocity fields we reduced it to a single one, instead of applying an update to fluid density only inside the fluid we now apply it everywhere,
we replaced the sharp interface if-cases switching between the multiple possible definitions of stress in the partial cells and now once again we want to define a single definition which mixes both stress and densities between the three phases and which can be applied field-wise.

The logic we applied to the mixing rule in order to preserve mass in the solid phases led us to the formulation of stress as in 4.31
Figure 4-27: Convergence plot for horizontal reference map component field $\xi_x$ for grid-sizes 60x60, 120x120, and 180x180 compared against a simulation with grid-size 360x360.

$$
\sigma_{11}[ije] = |\text{erf}f^1[ije] + \text{erf}f^2[ije] - 1| \times \sigma_{11}^{1xe}[ije] \\
+ (1 - \text{erf}f^1[ije]) \times \sigma_{11}^{1xe}[ije] \\
+ (1 - \text{erf}f^2[ije]) \times \sigma_{11}^{2xe}[ije]
$$

(4.31)

This is to say that the quantity of each of the solid error functions is always conserved and the rest up to a fraction of 1 is filled with fluid. Here, $\sigma^{1xe}$ corresponds to the stress obtained from reference map (and its extrapolation) for the first solid phase and $\sigma^{2xe}$ respectively for the second one. The same approach is applied in defining the density mix.
Figure 4-28: Convergence plot for horizontal reference map component derivative field $\frac{\partial \xi_x}{\partial x}$ for grid-sizes 60x60, 120x120, and 180x180 compared against a simulation with grid-size 360x360.

4.11.2 Setting contact conditions

Normally, given that we have a unique velocity field which is used to advect both solid reference maps it should be impossible to have penetration of one into the other. However, practice shows that this actually can happen if we force one into the other. The probable reason for a mechanism which allows this to happen is the fact that when advecting the reference map the part which is outside of the solid was extrapolated and can therefore contain errors. The result is that, if the two solids are pushed hard into each other their extrapolations may erroneously induce penetration and once this happens then there is no mechanism to take them apart as can be seen in 4-29.

In order to avoid this defect we need to set an additional condition on contact between solids. Fortunately, in our framework doing this is extremely straightforward
and will eventually be useful for setting various contact conditions. The quantity defined as the difference between the two solid interface functions (4.32) directly gives us the position of the mid-surface between the two solid boundaries as well as the distance to each solid and this plane at every time step and regardless of the topology of the two solids.

$$\phi_{12} = \phi_1 - \phi_2$$ \hspace{1cm} (4.32)

Whereas in a Lagrangian framework calculating this distance is typically a complicated process of solving a least squares problem between every node and every edge from the two solids in our framework it a simple subtraction. As you can see from the plot of the $\phi_{12}$ function in 4-30 the zero contour of the $\phi_{12}$ function lies exactly in between the two solids.

Now that we have this curve, we can define a Gaussian influence function that represents mutually repulsive forces which only act on the solids whenever the distance between the two solids is less than the width of the Gaussian curve. In other words the actual solid-solid contact is done through the inclusion of a repulsive body force in all solid points which overlap the Gaussian influence centered around the mid-surface (the red curve in fig. 4-30). The idea would then be use this narrow Gaussian bell respectively with the correct sign in each solid and only apply an extra body force
Figure 4-30: The difference function $\phi_{12}$ between the two levelsets giving us the mid-surface between the two solids

We thus define the Gaussian function as in 4.33. It is a function of $dx$ and it is also truncated at 0.01 to be sure that it only affects the solids inside the blur zone and does not do anything when the two solids are far apart from each other.

$$gaussian = \begin{cases} \frac{\exp(-\phi_{12}^2)}{dx/16} - 0.01, & \frac{\exp(-\phi_{12}^2)}{dx/16} - 0.01 > 0 \\ 0, & \frac{\exp(-\phi_{12}^2)}{dx/16} - 0.01 \leq 0. \end{cases}$$ (4.33)

Looking at 4-31 you can see how the position of the mid-plane is recalculated at every time step and how the solids do not feel the interaction until the moment when the narrow Gaussian function enters into the solids. This way of setting the
Figure 4-31: Zoom in on the Gaussian function (colored contours) centered around the mid-surface to correctly define contact and avoid penetration

contact condition acts as an external force which penalizes overlap and avoids it to ever happen. Other approaches also exist, we can for example measure the amount of overlap a step would cause and if any then apply a correction based on the intersecting volume [14] and a minimization problem finding the optimal velocity field which solves the equilibrium condition while minimizing overlap.

4.11.3 Results

This approach allows us to model frictionless non-sticking contact that avoids cross-penetration of the solid phases. Furthermore, it gives a framework for setting various contact conditions later on. We could imagine sticking conditions, friction or for
that matter any constitutive relation for the contact through new definitions of the influence function centered at the mid-surface. As you can see in 4-32 using this technique we are now capable of getting very large deformations in the solids through contact and without any problem of penetration or sticking of one solid onto the other.

Figure 4-32: Example of contact between an anchored and a mobile ball showing large deformations under contact immersed in a fluid

Another example we have simulation is a ball colliding with a stick anchored at one end and bouncing back from it. As can be seen in 4-33, 4-34 and 4-35 the ball hits the stick, deforms it to very large strain, bounces back off from it, and then falls back again through the action of a small amount of gravity.
Figure 4-33: Simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid
Figure 4-34: Simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid (part two)
Figure 4-35: Final frame of the simulation of pressure in the case of a fully coupled FSSI between a ball and a stick immersed in liquid.
Chapter 5

Conclusions

We have shown in this work a new and fully functional numerical method for the simulation of a fully-coupled fluid-solid-solid interaction (FSSI) or in that matter any number of solids interacting with each other all submerged in a fluid. We have followed the logical chain taking us from the initial proof-of-concept formulation which was based on a sharp-interface technique to a newer, simplified, more efficient, less numerically greedy and more stable implementation based on a blurred interface technique. We have analyzed various problems related to the extrapolation routine and the excessive application of the last. This work has also shown how to cope with a problem with an unknown origin by dissecting a numerical method to its bits and pieces in order to locate the initial occurrence of faults. We have thus found a few sources of error such as numerical artifacts in extrapolations, mismatch between the interface locus as defined per the level-set and per the reference map, checkerboarding patterns in stress fields and sudden “identity changes” of grid-cells because of the Eulerian fixed-grid approach.

5.1 Summary

The Reference Map Technique is a new method presented in 2012 by Kamrin [12] allowing for large deformation simulations of any thermodynamically admissible constitutive solid model on a fixed Eulerian grid. Combined with a levelset to track the
interfaces between phases we have at first presented a sharp interface implementation, showing the capabilities of the method to do fully-coupled FSI. In this work we have then analyzed previous problems with the sharp method and as a solution we have provided a blurred-interface implementation which also allowed to go a step beyond and simulate two solid phases immersed in a fluid where all interactions fully coupled in both solid-solid contact and in fluid-solid interactions.

5.2 Future work

As this method is very recent, most of the work has happened over the last few years there is a vast field of new possibilities which we could explore. The solid-solid interaction has already been implemented in 3D by Levin [14], although using a very different contact routine than the one we have discussed. The full FSI and FSSI codes have so far only been implemented in 2D, however, there is no limitation into porting the exact current approach into 3D and this is definitely one of the items on our short term to-do list. Solving multi-physics and multi-scale problems is also much simpler on a fixed mesh and adding an extra physical mechanism is as easy as adding an a extra field to store additional information and informing the code on how to update it in various phases. For instance, work on plasticity has already been done using the RMT where the deformation gradient is decomposed into an elastic and a plastic part and the second one is stored as an additional variable [11]. Very promising work is also currently being done in Prof. Kamrin’s group into sub-grid modeling in Eulerian schemes. As no mesh needs to ever be moved or deformed, adding in new points only when and only where we need them is much more easily done than it is in an equivalent Lagrangian scheme. Another direction of work for the short-term future is parallelizing the code and also running it on GPU. Once again, there is no actual barrier in doing so and it will eventually be done. Another direction for future will also be the extension to incompressible materials by utilizing the projection method as presented by Chorin [5]. Enforcing incompressibility in an Eulerian scheme is a linear constraint applied to the velocity field which is much simpler than the non-
linear constraint to the determinant of the deformation gradient $F$ as required in a Lagrangian frame. ☐
Appendix A

The corrected extrapolation routine for matching boundary conditions

A.1 Motivation

We have seen throughout this document how important the extrapolation steps are for basically any Eulerian method simulating a solid phase. Due to the inherent particularity in such methods that no points actually move we need to create values for solid fields outside of the solid boundary and thus create an initial guess as to what values we should have at these locations once the solid moves into them (from a Lagrangian point of view) or in other words once these grid cells become solid cells (from an Eulerian point of view). Whether, we are talking about a sharp interface method or a blurred interface one we eventually always require the extrapolation of at least one field. In the 2d implementation utilizing the sharp interface approach we used to require 13 scalar fields to be extrapolated, and in the blurred interface (sometime referred to as smeared interface) we required 2 scalar field extrapolations - the two components of the reference map vector field. From a conceptual point of view the idea is pretty simple - by extrapolating our values from inside the solid
we can easily populate the cells which reside outside of it with a guess. However, in practice this extrapolation is a numerical procedure and as any numerical procedure it has inherent sources of error. Due to the fact that the extrapolation is carried out on a discrete grid and we need to do extrapolations along the normal directions of the interface it is very quickly apparent that producing smooth results as one would expect from the analytical definition of our extrapolation scheme in the continuous limit is practically impossible. We have therefore come up with various steps to be applied post-extrapolation which would ensure that the so produced fields obey certain criteria required for our method. We have already discussed the post-extrapolation diffusion scheme in 4.8 and the scheme for removing fake surface tension forces is presented in appendix B. In the same sense, the new pinning scheme is an on-sight remedy against mismatch between the level-set and the extrapolated reference map (sec. 4.6).

A.2 The extrapolation correction routine ensuring boundary condition at the interface

One other such problem with the steps of extrapolation actually involves the fact that in the case of the sharp interface approach we do not only need to extrapolate values from the solid out into the fluid but similarly we need to extrapolate values from the fluid into the solid. These newly extrapolated fields should in addition meet some boundary conditions. One such example is solid velocity which is extrapolated in order to create the solid ghost velocities, similarly fluid ghost velocities are created and these two fields should obey a boundary condition on the interface. In the case of a no-slip boundary we would expect the tangential components of these two fields to meet at the same value right on the interface but due to numerical imprecisions this is not always the case (see fig.A-1). We have therefore, come up with a routine which is applied to the full fields once the extrapolations have been executed which finds the mismatch between any two given fields right at the interface and modifies
Figure A-1: Mismatch of two extrapolated fields at the interface due to numerical imperfections

them so that by construction they always agree to the same value on the interface. The scheme works based on the assumption that we only apply a correction very close to the interface. Since the extrapolation scheme creates a linear order extrapolation we can take the difference between the two fields which need to match up and then calculate its directional derivative along the normal of the interface. This gives us the slope right at the interface and gives us a way of removing the linear order term of the extrapolated values allowing us to calculate the mismatch \( c \) (fig.A-1) and remove

Figure A-2: Mismatch of two extrapolated fields at the interface due to numerical imperfections before and after the correction routine
the necessary amount from the two initial fields so that by construction they agree to the exact same value on the boundary.
**Data:** A solid field $f^s$ and a fluid field $f^f$ with their extrapolations which might not agree to the same value on the boundary

**Result:** The two fields modified close to the boundary so that they correctly agree on the same value on the boundary

1. Calculate the difference
   
   $d = f^s - f^f$;

2. By assumption it can be represented as a linear function of the distance to the boundary with a constant $c$ and a slope of $b$
   
   $d = c + b\phi(x) = c + \beta$

3. Evaluate the directional derivative of the $d$ field along normal directions of the levelset. The normal direction to the interface being given by the gradient of the $\phi$ field.
   
   $n_\perp \cdot \nabla d = \nabla \phi \cdot \nabla \cdot d$;

4. Evaluate the linear order term
   
   $\beta = b\phi(x) = (\nabla \phi \cdot \nabla \cdot d) \times \phi$;

5. Isolate only the mismatch constant $c$ from the $d$ field
   
   $c = d - \beta = d - (\nabla \phi \cdot \nabla \cdot d) \times \phi$;

6. Define correcting functions from the solid into the fluid and from the fluid into the solid
   
   $F_{sf} = (1 + \phi)^2$

   $F_{fs} = (1 - \phi)^2$;

7. Set the correct amplitude of the required correction at the interface
   
   $F_{sf} = c \times F_{sf}$

   $F_{fs} = c \times F_{fs}$;

8. Create the corrected difference fields (cd) for the two phases
   
   $cd_{sf} = d - F_{sf}$

   $cd_{fs} = d - F_{fs}$;

9. Apply the corrections to the initial fields and create the corrected $f^s$ and $f^f$
   
   $f^s_c = f^s + cd_{sf}/2$

   $f^f_c = f^f + cd_{fs}/2$;

**Algorithm 4:** Extrapolation correction routine ensuring boundary conditions
The correcting functions could be polynomial, cosine, Gaussian or any other function going through 1 right at the interface and going to 0 with a horizontal slope a few grid spacings away from the interface. By applying this step half of the correction required to remove the mismatch $c$ goes to the first field and the other half to the other field resulting in fields such as in fig.A-2.
Appendix B

Removing fake surface tension forces on curved boundaries

Figure B-1: Comparison of divergence of extrapolated stresses on a straight and on a curved interface

Various problems related to the extrapolation routines as well as just to the idea of extrapolating values regardless of the actual numerical schemes have been referenced throughout this work (appendix A, section 4.8). Another type of problem which can occur when relying on an extrapolation is the fact that even though a certain stress field satisfies equilibrium there is no reason why an extrapolated field based on it will also satisfy equilibrium. In most cases this should not necessarily pose a problem,
especially in the case of the blurred interface method where the extrapolated values for stresses (our guesses for the stress in newly acquired cells) are introduced gradually and therefore the dynamics of the simulation ensures that they satisfy all physical laws. However, in the case of the sharp interface implementation where stresses would get directly extrapolated we could describe a case in which the fact that we rely on extrapolated stresses could potentially create additional fake forces acting on the interface. Looking at fig.B-1 we can see how if we have a straight interface and a solid which is under pure tension the extrapolated values will also satisfy equilibrium. However, in the case of a curved interface (where curvature is high compared to the grid size - a case which should nevertheless be always avoided) the extrapolation of stresses would create a grid cell in which the divergence of stress term which updates velocity would actually create an additional normal fake body force, very similar to surface tension. We have therefore, come up with a scheme to quantify this amount of fake body force and remove it while calculating the divergence of stress term in the update to velocity.

Figure B-2: Calculating the fake body force created by extrapolating stress across a curved interface
Data: Solid stress field with its extrapolation

Result: Corrected divergence of stress calculation with removed fake body forces due to extrapolation across a curved interface

1. Define a staggered matrix field $\sigma^g_{tt}$

   $\sigma^g_{tt} = \begin{cases} 
   0, & \text{point } 1 \text{ is inside the solid phase} \\
   \sigma^g_{tt}(pt.1)n_\perp, & \text{the point is outside the solid phase.}
   \end{cases}$

   Where we have defined the staggered matrix field $n_\perp(pt.1) = [\nabla \phi_\perp]^T[\nabla \phi_\perp]$

2. The divergence of $\sigma^g_{tt}$ gives us a vector field of net body force

   $\vec{f}_{\text{net,tt}} = \nabla \cdot \sigma^g_{tt}$

3. Create the correction vector field isolating only the normal component

   $\vec{f}_{\text{fake}} = -[\vec{f}_{\text{net,tt}} \cdot \nabla \phi] \nabla \phi$

4. When calculating the divergence of stress use the new definition instead

   $\nabla \cdot \sigma := \nabla \cdot \sigma + \vec{f}_{\text{fake}}$

Algorithm 5: Removing fake body forces due to stress extrapolation scheme

By calculating the divergence of stress only considering points which are inside the solid we can quantify what part of the full divergence of stress considering the extrapolated values actually creates a fake normal force which should be removed. It is nevertheless important to point out that this effect even though not removed would get decreased by using a higher resolution in the simulation thus effectively reducing the curvature of the interface when compared to the grid size.
Appendix C

Rmapsim.cc source code

In the current implementation of the code the main file solving for the update from one time step to the next one is given in this appendix. The structure of this file follows the idea behind section 4.3.5. A fundamental tool used throughout this work is Chris Rycroft’s level-set library (for more information see [18]).

```cpp
#include "rmap_sim.hh"
#include "math.h"

/** Initializes a reference map simulation, allocating memory, setting up the
 * fields, and building the level set narrow band.
 * \param[in] sim_t_ the simulation type.
 * \param[in] (m_,n_) the number of gridpoints in the horizontal and vertical
 * \param[in] (ax_,bx_) the lower and upper simulation bounds in the horizontal
 * \param[in] (ay_,by_) the lower and upper simulation bounds in the vertical
 * \param[in] tmult_ the multiplier to apply to the default timestep computation.
 * \param[in] ntrace_ the number of fluid tracers to initialize. If zero is given
 * \param[in] filename_ the output file */
rmap_sim::rmap_sim(sim_type sim_t_,int m_,int n_,double ax_,double bx_,
                  double ay_,double by_,
                  double tmult_,const int ntrace_,const char* filename_)
  : k_total(0), k_global(0), sim_t(sim_t_), m(m_), n(n_), mn(m_*n_), me(m_)
```
slixef(new double[mne]), corrr(new double[mn]), navg(new double[mn]),
erff(new double[mne]),
rhom(new double[mn]), crhof(new double[mn]), crhof1(new double[mn]),
crhof2(new double[mn]),
erffr(new double[mn]), rhos(new double[mn]), crhos(new double[mn])
{
    // Set up the initial fields
    init_fields();

    // Build the narrow-banded level set
    ls.build_band();

    // Initialize the tracers
    if(ntrace>O) initialize_tracers();

    // Set up the staggered indicator array
    for(int *ccp=cc;ccp<cc+mne;) *(ccp++)=7;
}

/** The class destructor frees the dynamically allocated memory. */
rmap_sim::~rmap_sim()
{
    delete [] buf;if(ntrace>0) delete [] tp;
    delete [] cy;delete [] cX;delete [] cv;delete [] cu;
    delete [] rhof;delete [] s22f;delete [] s12f;delete [] s11f;
    delete [] rho;delete [] s22;delete [] s12;delete [] s11;
    delete [] vf;delete [] uf;
    delete [] Y;delete [] X;
    delete [] v;delete [] u;
    delete [] un; delete [] un2; delete [] un3;
    delete [] un4; delete [] cun;

delete [] cu1; delete [] cu2; delete [] cu3;
    delete [] cv1; delete [] cv2; delete [] cv3;
    delete [] bfdel; delete [] s11m; delete [] s12m;
    delete [] s22m; delete [] JJ;
    delete [] Xxm; delete [] Xym;
    delete [] Yxm; delete [] Yym; delete [] tempe;
    delete [] F1; delete [] F2;
    delete [] F3; delete [] F4;
    delete [] s11xe; delete [] s12xe; delete [] s22xe;
    delete [] s11p; delete [] s12p; delete [] s22p;

delete [] l1; delete [] ccc;
    delete [] Xt; delete [] Yt;
    delete [] ufnx, delete [] vfnx;
    delete [] s11r; delete [] corr;
    delete [] s11xef; delete [] corrr;
/** Sets up the fluid tracers by initializing them at random positions within
 * the fluid phase. */
void rmap_sim::initialize_tracers() {
    double tx,ty,phix,phiy;
    for(double *tps=tp; tps<tp+(ntrace<<1);) {

        // Create a random position vector within the simulation region
        tx=ax+m*dx*rnd();
        ty=ay+n*dy*rnd();

        // If this position vector is within the fluid, then store it.
        // Otherwise, skip it.
        if(ls.bicubic(tx,ty,phix,phiy)>0) {
            *(tps++)=tx; *(tps++)=ty;
        }
    }
}

// Moves the tracers according to the extrapolated fluid velocity.
// \param[in] dt the timestep to use. */
void rmap_sim::update_tracers(double dt) {
    double x,y,*tps=tp,*tpe=tp+(ntrace<<1);
    while(tps!=tpe) {

        // Read the tracer position, compute its updated position

        }
```c
x=tps;y=*(tps+1);
x+=dt*ls.bicubic(uf,x,y);
y+=dt*ls.bicubic(vf,x,y);

// Deal with cases where the tracers may flow out of the simulation and many need to be reinserted
if(sim_t==rotor&&y>ay+n*dy) {
    x=ax+y-ay-n*dy;
    y=ay+n*dy*rnd()*0.5;
} else if(sim_t==stick&&x>ax+m*dx) {
    x-=m*dx;
    y=ay+n*dy*rnd();
}

// Store the updated position
*(tps++)=x;*(tps++)=y;

// In the rotor simulation, also move the rotor mark
if(sim_t==rotor) {
    x=dt*ls.bicubic(uf,rmarkx,rmarky);
    y=dt*ls.bicubic(vf,rmarkx,rmarky);
    rmarkx+=x;rmarky+=y;
}

/** Save the tracers to a file that can be read by GnuPlot. 
 * \param[in] sn the frame number to append to the filename. */
void rmap_sim::output_tracers(const int sn) {

    // Assemble the output filename and open the file
    char *bufc=((char*) buf);
    sprintf(bufc,"Xs/trace.%d",filename,sn);
    FILE *outp=safe_fopen(bufc,"w");

    // Output the tracers to file
    for(double *tps=tp;tps<tp+(ntrace<<1);tps+=2) fprintf(outp,"Xg %g
",*tps ,*(tps+1));

    // Close the output file
    fclose(outp);
}

void rmap_sim::output_vector_field_normal(const int sn,const double spc,
                          const double fac) {
    double s=1/spc,x,y;
```

int li=ax*s+1, lj=ay*s+1, ui=(ax+m*dx-1e-8)*s, uj=(ay+n*dy-1e-8)*s, i, j;
double n1, n2;

// Assemble the output filename and open the file
char *bufc=((char*) buf);
sprintf(bufc, "%s/vfld.%d", filename, sn);
FILE *outp=safe_fopen(bufc, "w");

// Loop over the vector field grid
for(j=lj; j<=uj; j++) {
  y=j*spc;
  for(i=li; i<=ui; i++) {
    x=i*spc;
    // nl=phi-x(i,ij); n2=phi-y(ij);

    // If this gridpoint is within the fluid phase then
    // output a vector
    // if(is.bicubic(phi,x,y)>dx)
    fprintf(outp, "Xg %g 0 %g %g
", i*spc, j*spc, fac*ls.bicubic(nnl, x, y), fac*ls.bicubic(nn2, x, y));
  }
}

// Close the output file
fclose(outp);

/** Output the fluid velocity as a vector field that can be read by Gnuplot
.*
.* \param[in] sn the frame number to append to the filename.
.* \param[in] spc the spacing to be used between each vector.
.* \param[in] fac the multiplier to be used to compute the vector length.
.* */
void rmap_sim::output_vector_field(const int sn, const double spc, const
double fac) {
  double s=1/spc, x, y;
  int li=ax*s+1, lj=ay*s+1, ui=(ax+m*dx-1e-8)*s, uj=(ay+n*dy-1e-8)*s, i, j;

  // Assemble the output filename and open the file
  char *bufc=((char*) buf);
  sprintf(bufc, "%s/vfld.%d", filename, sn);
  FILE *outp=safe_fopen(bufc, "w");

  // Loop over the vector field grid
  for(j=lj; j<=uj; j++) {
    y=j*spc;
for(i=li;i<=ui;i++) {
    x=i*spc;

    // If this gridpoint is within the fluid phase then
    // output a vector
    //if (is.bicubic(phi, x, y) > dx)
    fprintf(outp, "%g %g 0 %g %g 0\n",
            i*spc, j*spc, fac*ls.bicubic(uf, x, y), fac*ls.bicubic(vf, x, y));
}

// Close the output file
fclose(outp);
}

/** Carries out reference map simulation.
 * \param[in] (t_start, t_end) the time interval to simulate over.
 * \param[in] frames the number of output frames to store. */
void rmap.sim::solve(double t_start, double t_end, int frames) {
    double time = t_start, time_interval = (t_end - t_start) / frames, target_time, dt =
    dx*dx*tmult;
    int k;

    // Output header information for movie annotation
    char *bufc = ((char*) buf);
    sprintf(bufc, "%s/header", filename);
    FILE *fh = safe_fopen(bufc, "w");
    fprintf(fh, "%g %g %d\n", t_start, t_end, frames);
    fclose(fh);

    // Set up the rotor mark file for the rotor simulation
    if(sim_t == rotor) {
        sprintf(bufc, "%s/rmark", filename);
        rmarkf = safe_fopen(bufc, "w");
        rmarkx = 0; rmarky = 1;
    }

    // Carry out a preliminary regular field extrapolation and output the
    // initial frame

    extrapolate_regular();
    printf("before first compute stress\n");
    compute_stress();
    write_files(k_total);
    printf("=============Output frame 0, t=%g =========\n", time);

    // Loop over the frames

for (k = k_total + 1; k <= k_total + frames; k++) {
    // Calculate the time to the next frame
    target_time = t_start + time_interval * k;
    current_frame = k;

    // Keep stepping forward with the regular
    while (time + dt + 1e-11 < target_time) {
        step_forward(dt);
        time += dt;
        //printf("step without output- k=%d, time=%g, dt=%g\n",k,time,dt);
    }

    step_forward(target_time - time);
    time = target_time;

    // Print a status message and save snapshots of the fields
    printf("==============Output frame %d, t=%g ==========
", k, time);
    write_files(k);
    k_global = k;
}

k_total += frames;
// Close the rotor mark file if it is in use
if (sim_t == rotor) fclose(rmarkf);
}

/** Advances the simulation fields forward by one timestep */
* \param[in] dt the timestep to use. */
void rmap_sim::step_forward(double dt) {
    // Reference map update
    reference_map_update(dt);

    // Calculate the error functions
    calculate_fraction();

    // Compute solid stress
    compute_stress();

    // Add viscosity terms and mixes
    fix_boundary();

    // Compute next timestep for solid fields
    calculate_field_update(dt);

    // Update tracer positions
    if (ntrace > 0) update_tracers(dt);
// Update solid fields
update_fields();

// Apply boundary conditions
anchor();

// Extrapolate fields
extrapolate_regular();
}

/**
  * Carries out extrapolation of all of the regular non-staggered fields.
  */
void rmap_sim::extrapolate_regular() {

    // Extrapolate solid fields
    ls.set_fields(X,Y);
    ls.extrapolate_fields();

    tangential_diffusion(X);
    tangential_diffusion(Y);
}

/**
  * Updates the reference map fields based on the current velocity.
  * \param[in] dt the timestep to update by. */
void rmap_sim::reference_map_update(double dt) {
    int i,j,ij,ck;
    double X1,Y1,X2,Y2,hx=0.5*dt*xsp,hy=0.5*dt*ysp;
    double phix,phiy,yyy,xxx,phhh,s;
    const double prop_1=1.1;
    const double prop_w=0.4;
    const double prop_b=0.8;

    // Loop over all of the gridpoints within the solid phase
    for(i=0;i<m;i++)
        for(j=0;j<n;j++) {

            if(c[ij]<4) {
                // Compute the derivatives in the horizontal direction
                if(uf[ij]>0) {
                    if(i>1&c[i-2]<4) {X1=hx*eno2(X,ij,-1);Y1=hx*eno2(Y,ij,-1);}
                    else {X1=2*hx*(X[ij]-X[ij-1]);Y1=2*hx*(Y[ij]-Y[ij-1]);}
                }
                else {
                    if(i<m-2&c[i+2]<4) {X1=-hx*eno2(X,ij,1);Y1=-hx*eno2(Y,ij,1);}
                }
            }

        }
}

111
else {X1=2*hx*(X[ij+1]-X[ij]);Y1=2*hx*(Y[ij+1]-Y[ij]);}
}

// Compute the derivatives in the vertical direction
if(vf[ij]>0) {
    if(j>1&&c[ij-2*m]<4) {X2=hy*eno2(X,ij,-m);Y2=hy*eno2(Y,ij,-m);}
    else {X2=2*hy*(X[ij]-X[ij-m]);Y2=2*hy*(Y[ij]-Y[ij-m]);}
} else {
    if(j<n-2&&c[ij+2*m]<4) {X2=-hy*eno2(X,ij,m);Y2=-hy*eno2(Y,ij,m);}
    else {X2=2*hy*(X[ij+m]-X[ij]);Y2=2*hy*(Y[ij+m]-Y[ij]);}
}

// Update the fields
cX[ij]=-uf[ij]*X1-vf[ij]*X2;
cY[ij]=-uf[ij]*Y1-vf[ij]*Y2;

}

// Run the pinning routine several times to pin phi to psi
for(ck=0;ck<10;ck++) {

// Loop over regular grid to obtain the speed field moving phi everywhere on the grid before calling the ls.move function
for(ij=j=Q;j<n;j++) for(i=Q;i<m;i++,ij++) {

// Initialize the psi interface based on the initial level set function
switch(sim_t) {
    case circle:
        xxx=0.8*X[ij];xxx+=0.5*Y[ij];
        yyy=Y[ij];
        psi[ij]=-0.8+sqrt(xxx*xxx+yyy*yyy);
        break;
    case rotor:
        xxx=abs(X[ij]);yyy=abs(Y[ij]);
        if(xxx<prop_b&&yyy<prop_b) {
            xxx=prop_b;
            yyy=prop_b;
            psi[ij]=prop_b-prop_w-sqrt(xxx*xxx+yyy*yyy);
        } else {
            if(xxx<yyy) {s=xxx;xxx=yyy;yyy=s;}
            yyy)-prop_w;
        }
        break;
    case stick:

yyy = abs(Y[ij]) - prop_1;
if (yyy < 0) yyy = 0;
xxx = X[ij];
phhh = sqrt(xxx*xxx + yyy*yyy) - prop_w;

psi[ij] = sqrt(xxx*xxx + yyy*yyy) - prop_w;

} //switch

// Define area to be corrected
if (phi[ij] > 5*dx) {
    dpsi[ij] = phi[ij] - psi[ij];
} else {
    dpsi[ij] = 0.0;
}

// Set difference field as velocity to move phi to psi
un3[ij] = un[ij] = 5*dpsi[ij];

} // Loop over regular grid

ls.move(this, dt);

}

void rmap_sim::construct_normal_field(double *f1, double *f2, double *f) {
    for(int ij=0; ij < mn; ij++) {
        f[ij] = f1[ij]*nn1[ij] + f2[ij]*nn2[ij];
    }
}

void rmap_sim::construct_normal_field() {
    int i, j, ij;

    for (ij = j = 0; j < n; j++) for (i = 0; i < m; i++, ij++) {
        nn1[ij] = phi_x(i, ij);
        nn2[ij] = phi_y(ij);
    }
}

/** Normalizes a two-dimensional position vector to have length 1.
 * \param[in,out] (vl,v2) the vector coordinates. */
inline bool rmap_sim::normalize(double &vl, double &v2) {
    double mag = v1*v1 + v2*v2;
    if (mag > 1e-16) {
        mag = 1.0/sqrt(mag);
/** Computes the fluid and solid stress tensors. */
void rmap_sim::computestress()
{
  const double rho0=1,rhof0=1,rhof0_inv=1/rhof0,lambdaf=1;
  int ie,je,ije,k,1,noex;
  double Xx,Yx,Xy,Yy,pf,alpha,beta,J,Jinv;
  double u1,v1,rho1,u2,v2,rho2,u3,v3,rho3,u4,v4,rho4;
  double xp,yp;
  double n1,n2;
  double sp11,sp12,sp22;
  mat F,FT,sigma,B;

  // Constants used in the stress model. The 3D shear modulus G is equal
  // to the 2D shear modulus. The 2D bulk modulus kappa differs from the
  // full 3D one by a correction term.
  const double fac=5/3.,third=1/3.,G=10,kappa=10; //was 50

  // Loop over all gridcells
  for(ije=je=0;je<=n;je++) for(ie=0;ie<=m;ie++,ije++)
  {
    xp=ax + (ie+0.5)*dx;
    yp=ay + (je+0.5)*dy;

    ij=ie+m*je;

    // Count the number of surrounding gridpoints which are in the fluid
    k=1=0;
    if(ie>0) {
      if(je>0) {k++;if(c[ij-m-1]>3) l++;}
      if(je<n) {k++;if(c[ij-1]>3) l++;}
    }
    if(ie<m) {
      if(je>0) {k++;if(c[ij-m]>3) l++;}
      if(je<n) {k++;if(c[ij]>3) l++;}
    }

    // Compute solid stress everywhere
    if(ccc[ije]<6)
    {
      //Old stencil for rmap gradient
      Xx=0.5* (derivx(X,ij) + derivx(X,ij-m));
      Xy=0.5* (derivy(X,ij) + derivy(X,ij-1));
Yx = 0.5 * (derivx(Y, ij) + derivx(Y, ij-m));
Yy = 0.5 * (derivy(Y, ij) + derivy(Y, ij-1));

// New triangular stencil for rmap
Xx = 1.0 * (O*derivx(X, ij) + derivx(X, ij-m));
Xy = 1.0 * (O*derivy(X, ij) + derivy(X, ij-1));
Yx = 1.0 * (O*derivx(Y, ij) + derivx(Y, ij-m));
Yy = 1.0 * (O*derivy(Y, ij) + derivy(Y, ij-1));

J = 1 / (Jinv = Xx*Yy - Xy*Yx);
Xxm[ije] = Xx;
Xym[ije] = Xy;
Yxm[ije] = Yx;
Yym[ije] = Yy;
JJ[ije] = J;
F1[ije] = Yym[ije]*JJ[ije];
F2[ije] = -Xym[ije]*JJ[ije];
F3[ije] = -Yxm[ije]*JJ[ije];
F4[ije] = Xxm[ije]*JJ[ije];

J = JJ[ije];
Jinv = Xxm[ije]*Yym[ije] - Xym[ije]*Yxm[ije];
F = mat(F1[ije], F2[ije], F3[ije], F4[ije]);
FT = transpose(F);
B = F*FT;

// Compressible Neo-Hookean model.
// Strain-Energy: W = 0.5*G*(I_1-3)+0.5*kappa*(J-1)^2
// G is small-strain shear modulus and kappa small-strain bulk modulus.
// J = det F and I_1 = J^(-2/3) trace(B)
// This model is applied under plane-strain conditions.
alpha = pow(Jinv, fac) * G;
beta = kappa * (J-1) - (G * pow(Jinv, fac)) * (1+trace(B)) * third;
sigma = alpha*B + beta*mat(1, 0, 0, 1);
rho[ije] = rho0*Jinv;

// Store the computed stress
s11xe[ije] = sigma.a;
s12xe[ije] = sigma.b;
s22xe[ije] = sigma.d;

// Compute fluid stress at solid gridpoints
ghost_fluid(ie, je, ij, u1, v1, rho1);
ghost fluid(ie,je-1,ij-m,u2,v2,rho2);
ghost fluid(ie-1,je,ij-1,u3,v3,rho3);
ghost fluid(ie-1,je-1,ij-m-1,u4,v4,rho4);

// Store the stress
pf=lamda*(rho4)*rhof0_inv-1);
s11e[ije]=-pf;
s12e[ije]=0;
s22e[ije]=0;
}

} else {
  rho[ije]=rho0;
s11x[ije]=0.0;
s12x[ije]=0.0;
s22x[ije]=0.0;
}

// Set correct cc values and compute fluid stress values
if(l==0) {
  // If all of the nearby gridpoints are in the solid
  // phase, then mark this gridpoint as a "full solid"
  // gridpoint
  cc[ije]=0;
}

} else if(l==k) {

    // If all of the nearby gridpoints are in the fluid
    // phase, then mark this gridpoint as a "full fluid"
    // gridpoint
    cc[ije]=7;

    // Calculate the density in order to
    ghost fluid(ie,je,ij,u1,v1,rho1);
    ghost fluid(ie,je-1,ij-m,u2,v2,rho2);
    ghost fluid(ie-1,je,ij-1,u3,v3,rho3);
    ghost fluid(ie-1,je-1,ij-m-1,u4,v4,rho4);

    // Store the stress
    //pf=lamda*(0.25*(rho1+rho2+rho3+rho4)*rhof0_inv-1);
    pf=lamda*((rho4)*rhof0_inv-1);
s11f[ije]=-pf;
s12f[ije]=0;
s22f[ije]=-pf;

    s11m[ije]=s11f[ije];
s12m[ije]=s12f[ije];
s22m[ije]=s22f[ije];

} else {
cc[ije]=staggered_phi(ie,je,ij)<0?3:4;

// Calculate the density in order to
ghost_fluid(ie,je,ij,u1,v1,rhol);
ghost_fluid(ie,je-1,ij-m,u2,v2,rho2);
ghost_fluid(ie-1,je,ij-1,u3,v3,rho3);
ghost_fluid(ie-1,je-1,ij-m-1,u4,v4,rho4);

// Store the stress
//pf=lambdaf*(0.25*(rhol+rho2+rho3+rho4)*rhof0_inv-1);
pf=lambdaf*((rho4)*rhof0_inv-1);
s11f[ije]=-pf;
s12f[ije]=0;
s22f[ije]=-pf;

s11m[ije]=s11f[ije];
s12m[ije]=s12f[ije];
s22m[ije]=s22f[ije];
}

} // close loop over gridcells

} // close compute_stress

void rmap_sim::tangential_diffusion(double *f1) {
    int ij,i,j; // navg;
    double n1,n2;
    double Df,dtf;
    double dxx,dxy,dyy;
    int ptt,tt;
    double dxx_max, dxy_max,dyy_max,dmax;
    double dxx_maxe,dxy_maxe,dyy_maxe;
    ptt = 0;

    //find max curvature inside the solid
    dxx_max=dxy_max=dyy_max=0.0;
    for(ij=j=0;j<0*n;j++) for(i=0;i<m;i++,ij++) {
        //Get normal direction
        n1 = phi_x(i,ij);
        n2 = phi_y(ij);

        //...
if (c[ij]<4) {
  dxx = (f1[ij+1]-2*f1[ij]+f1[ij-1])*xsp*xsp;
  dxy = (f1[ij+m+1]-f1[ij+1-m]-f1[ij-1+m]+f1[ij-m+1])*xsp*ysp*0.25;
  dyy = (f1[ij+m]-2*f1[ij]+f1[ij-m])*ysp*ysp;

  if (abs(dxx) > dxx_max) dxx_max = abs(dxx);
  if (abs(dxy) > dxy_max) dxy_max = abs(dxy);
  if (abs(dyy) > dyy_max) dyy_max = abs(dyy);
}
}
do {
  dxx_maxe=dxy_maxe=dyy_maxe=0;

  for(ij=j=0;j<n;j++) for(i=0;i<m;i++,ij++) {
    //staggered_grad_phi(n1,n2,ie,je);
    n1 = nn1[ij];
    n2 = nn2[ij];

    if ( (ij-m-1>0 && c[ij]>=3 && c[ij]<=6 && c[ij+m+1]<=6 && c[ij-m+1]<=6 && c[ij-m]<=6 && c[ij+m]<=6 && c[ij+m-1]<=6 && c[ij+1]<=6 && c[ij-1]<=6) ) {
      dxx = (f1[ij+1]-2*f1[ij]+f1[ij-1])*xsp*xsp;
      dxy = (f1[ij+m+1]-f1[ij+1-m]-f1[ij-1+m]+f1[ij-m+1])*xsp*ysp*0.25;
      dyy = (f1[ij+m]-2*f1[ij]+f1[ij-m])*ysp*ysp;

      //Only do tangential diffusion
      //corrr[ij] = dxx * n2*n2 -
      //  2*dxy*n1*n2 +
      //  dyy*n1*n1;

      //Do uniform diffusion
      corrr[ij] = dxx + dyy;

      Xt[ij] = f1[ij] + 0.000025*1*corrr[ij];

if (abs(dxx) > dxx_maxe) dxx_maxe = abs(dxx);
if (abs(dxy) > dxy_maxe) dxy_maxe = abs(dxy);
if (abs(dyy) > dyymaxe) dyy_maxe = abs(dyy);
} else {
  Xt[ij] = f1[ij];
}

for(ij=j=0; j<n; j++)
  for(i=0; i<m; i++, ij++) {
    if (c[ij] > 3 && phi[ij]<5*dx) f1[ij] = Xt[ij];
  }

ptt++;
//Define a criterion based on the curvature of the field inside the
//solid or if never met end after a fixed number of iterations

// } while ((dxx_maxe>0.25*dxx_max || dxy_maxe>0.25*dxy_max || dyy_maxe
//>0.25*dyy_maxe)&&(ptt<100));
} while ((1||dxx_maxe>0.25*dxx_max || dxy_maxe>0.25*dxy_max)&&(ptt<80));
//printf("tangential diffusion %d \n", ptt);

}

void rmap_sim::calculate_fraction() {
  int ije,je,ie,ij, nb;
  double A,An,Ad,Frac;
  double ph,phr;

  for(ije=je=0; je<ne; je++)
    for(ie=0; ie<me; ie++, ije++) {
      ij=ie+m*je;

      ph=staggered_phi(ie,je,ij)/(2*dx);
      phr=phi[ij]/(2*dx);

      //Set A, A numerator, A denominator, and Frac to zero
      A=An=Ad=Frac=0.0;

      //Remember to check what should happen when interface is at edge!
      if (ie>0) {
        if(je>0) {An+=phi[ij-m-1]; Ad+=abs(phi[ij-m-1]);}
        if(je<n) {An+=phi[ij-1]; Ad+=abs(phi[ij-1]);}
      }
      if(ie<m) {
        if(je>0) {An+=phi[ij-m]; Ad+=abs(phi[ij-m]);}
      }
  
  //Remember to check what should happen when interface is at edge!
  if (je>0) {
    if(ie>0) {An+=phi[ij-m-1]; Ad+=abs(phi[ij-m-1]);}
    if(ie<n) {An+=phi[ij-1]; Ad+=abs(phi[ij-1]);}
  }
  if(je<m) {
    if(ie>0) {An+=phi[ij-m]; Ad+=abs(phi[ij-m]);}
  }
if(je<n) {An+=phi[ij]; Ad+=abs(phi[ij]);}
}

A = An/Ad;

//Calculate the fraction of fluid in any cell
//0 - full solid; 1 - full fluid
Frac[ije] = Frac = 0.5*(1+A);

erff[ije] = (1+erf(ph))/2;
erffr[ij] = (1+erf(phr))/2;

//Define what the width of the narrow band is in order to set ccc field
nb = 6; //dx

if (Frac[ije] > 0.5) {
    ccc[ije] = (staggered_phi(ie,je,ij)<(nb-2)*dx)?5:
              (staggered_phi(ie,je,ij)<nb*dx)?6:7;
} else {
    ccc[ije] = (staggered_phi(ie,je,ij)>-(nb-2)*dx)?2:
              (staggered_phi(ie,je,ij)>-nb*dx)?1:0;
}

/** Applies the boundary conditions to the stress tensors at the fluid solid
 * interface. */
void rmapsim::fixboundary() {
    const double visc=0.12,s_visc=0.0;
    int ie,je,i,ij;
    double dsigmatt,u1,u2,u3,u4,v1,v2,v3,v4;
    double ux,uy,vx,vy,rhol,rho2,rho3,rho4;
    double A,An,Ad,Frac,sp11,sp12,sp22;

    // Loop over the staggered gridpoints
    for(ije=je=0;je<ne;je++) for(ie=0;ie<me;ie++,ije++) {
        ij=ie+m*je;

        // If this is not a full solid gridpoint, then add in fluid
        // stress components for regular fluid viscosity
        //if(cc[ije]!=0) {
ghost_fluid(ie,je,ij,u1,v1,rho1);
ghost_fluid(ie,je-1,ij-m,u2,v2,rho2);
ghost_fluid(ie-1,je,ij-1,u3,v3,rho3);
ghost_fluid(ie-1,je-1,ij-m-1,u4,v4,rho4);

ux=1*xsp*(0*u1+u2-0*u3-u4);
vx=1*xsp*(0*v1+v2-0*v3-v4);
uy=1*ysp*(0*u1-0*u2+u3-u4);
vy=1*ysp*(0*v1-0*v2+v3-v4);

tempre[ije]=uy;
s11f[ije]+=visc*ux;
s12f[ije]+=0.5*visc*(uy+vx);
s22f[ije]+=visc*vy;

if (cc[ije] != 7) {
    ux=1.0*xsp*(0*u[ij]-0*u[ij-1]+uf[ij-m]-uf[ij-m-1]);
    uy=1.0*ysp*(0*u[ij]+0*u[ij-1]-uf[ij-m]-uf[ij-m-1]);
    vx=1.0*xsp*(0*v[ij]-0*v[ij-1]+vf[ij-m]-vf[ij-m-1]);
    vy=1.0*ysp*(0*v[ij]+0*v[ij-1]-vf[ij-m]-vf[ij-m-1]);

    s11[ije]+=s_viscc*ux;
    s12[ije]+=0.5*s_viscc*(uy+vx);
    s22[ije]+=s_viscc*vy;
}

//Mix stress components
sp1=erff[ije]*s11f[ije] + (1-erff[ije])*s11xe[ije];
sp2=erff[ije]*s12f[ije] + (1-erff[ije])*s12xe[ije];
sp22=erff[ije]*s22f[ije] + (1-erff[ije])*s22xe[ije];

//Mix densities
rhom[ije] = erfr[ije]*rho[ije] +
(1-erfr[ije])*(rho[ije]+rho[ije+1]+rho[ije+me]+rho[ije+me+1])/4;

//Replace stress components with the mixed values
s11[ije] = s11f[ije] = sp1;
s12[ije] = s12f[ije] = sp2;
s22[ije] = s22f[ije] = sp22;
**Evaluates the level set at a staggered gridpoint. In the interior of the grid, the value is given as the average of four adjacent values. At boundary points, it uses linear extrapolation the closest four available values.**

*param[in] (ie,je) the coordinates of the staggered gridpoint.
*param[in] ij the index of the regular grid point to the upper right of the given gridpoint.
*return The computed level set value. */

```cpp
inline double rmap_sim::staggered_phi(int ie, int je, int ij) {
    int di = ie == 0 ? 1 : (ie == m ? -1 : 0), dj = je == 0 ? 1 : (je == n ? -1 : 0);
    if (di == 0 && dj == 0)
        return 0.25 * (phi[ij - m - 1] + phi[ij - m] + phi[ij - 1] + phi[ij]);
    ij += di + m * dj;
    return (0.5 + dj) * ((0.5 + di) * phi[ij - m - 1] + (0.5 - di) * phi[ij - m])
            + (0.5 - dj) * ((0.5 + di) * phi[ij - 1] + (0.5 - di) * phi[ij]);
}
```

**Evaluates the derivative of the level set function at a staggered gridpoint.**

*param[out] (phix,phiy) the derivative.
*param[in] (ie,je) the coordinates of the staggered gridpoint. */

```cpp
inline void rmap_sim::staggered_grad_phi(double &phix, double &phiy, int ie, int je) {
    double *phip = phi + (ie == 0 ? 0 : (ie == m ? m - 2 : ie - 1)) + m * (je == 0 ? 0 : (je == n ? n - 2 : je - 1));
    phix = 0.5 * xsp * (phip[m + 1] + phip[1] - phip[m] - *phip);
    phiy = 0.5 * ysp * (phip[m + 1] - phip[1] + phip[m] - *phip);
}
```

// Calculates the update to the velocity fields and fluid density field
// \param[in] dt the timestep to use. */

```cpp
void rmap_sim::calculate_field_update(double dt) {
    const double ex_visc = 0.02; // 0.03 0.06
    double hx = 0.5 * dt * xsp, hy = 0.5 * dt * ysp;
    double hxx = 2 * hx * xsp, hyy = 2 * hy * ysp, uxx, vxx;
    int i, j, ij, ije;
    double u1, v1, u2, v2, rhof1, rhof2;
    double s11x, s12x, s12y, s22y, iden, ufx, vfy;
    double ud, vd, rhod, uu, vu, rhou, phix, phiy, bfdelta;
    double rhos1, rhos2;
```

122
for(ije=ij=j=0;j<n;j++,ije++) for(i=0;i<m;i++,ij++,ije++) {

    // Compute one-sided derivatives in the x direction
    ghost_fluid(i-1,j,ij-1,ud,vd,rhod);
    ghost_fluid(i+1,j,ij+1,uu,vu,rhou);
    if(uf[ij]>0) {
        if((ij-2)>0&&i>1&&c[ij-2]<7) {
            u1=hx*eno2(uf,ij,-1);
            v1=hx*eno2(vf,ij,-1);
            rhofl=hx*eno2(rhof,ij,-1);
        } else {
            u1=2*hx*(uf[ij]-ud);
            v1=2*hx*(vf[ij]-vd);
            rhofl=2*hx*(rhof[ij]-rhod);
        }
    } else {
        if((ij+2)<=m*n&&i<m-2&&c[ij+2]<7) {
            u1=-hx*eno2(uf,ij,1);
            v1=-hx*eno2(vf,ij,1);
            rhofl=-hx*eno2(rhof,ij,1);
        } else {
            u1=2*hx*(uu-uf[ij]);
            v1=2*hx*(vu-vf[ij]);
            rhofl=2*hx*(rhou-rhof[ij]);
        }
    }

    // Evaluate x derivatives of fluid stress components
    s1lx=hx*(s1lf[ije+me+1]-s1lf[ije+me]+s1lf[ije+1]-s1lf[ije]);
    s1lx=2*hx*(s1lf[ije+me+1]-s1lf[ije+me]+0*s1lf[ije+1]-0*s1lf[ije]);
    s12x=hx*(s12f[ije+me+1]-s12f[ije+me]+s12f[ije+1]-s12f[ije]);
    s12x=2*hx*(s12f[ije+me+1]-s12f[ije+me]+0*s12f[ije+1]-0*s12f[ije]);

    // Evaluate fluid divergence term
    ufx=2*hx*(uu-ud); 

    // Evaluate additional x viscosity term
    uxx=hxx*(uu-2*uf[ij]+ud);
    vxx=hxx*(vu-2*vf[ij]+vd); 

    // Compute one-sided derivatives in the y direction
    ghost_fluid(i,j-1,ij-m,ud,vd,rhod);
    ghost_fluid(i,j+1,ij+m,uu,vu,rhou);
    if(vf[ij]>0) {
        if((ij-2*m)>0&&j>1&&c[ij-2*m]<7) {
            u2=hy*eno2(uf,ij,-m);
        } else {
            u2=2*hy*(uu-uf[ij]);
            v2=2*hy*(vu-vf[ij]);
            rhofl=2*hy*(rhou-rhof[ij]);
        }
    }
}

123
v2=hy*eno2(vf,ij,-m);
  rhof2=hy*eno2(rhof,ij,-m);
} else {
  u2=2*hy*(uf[ij]-ud);
  v2=2*hy*(vf[ij]-vd);
  rhof2=2*hy*(rhof[ij]-rhod);
}
} else {
  if((ij+2*m)<=m*n&&j<n-2&&c[ij+2*m]<7) {
    u2=-hy*eno2(uf,ij,m);
    v2=-hy*eno2(vf,ij,m);
    rhof2=-hy*eno2(rhof,ij,m);
  } else {
    u2=2*hy*(uu-uf[ij]);
    v2=2*hy*(vu-vf[ij]);
    rhof2=2*hy*(rhou-rhof[ij]);
  }
}

// Compute y derivatives of solid stress components
s12y=hy*(s12f[ije+me+1]+s12f[ije+me]-s12f[ije+1]-s12f[ije]);
s22y=hy*(s22f[ije+me+1]+s22f[ije+me]-s22f[ije+1]-s22f[ije]);
s12y=2*hy*(s12f[ije+me+1]+0*s12f[ije+me]-s12f[ije+1]-0*s12f[ije]);
s22y=2*hy*(s22f[ije+me+1]+0*s22f[ije+me]-s22f[ije+1]-0*s22f[ije]);

// Evaluate fluid divergence term
vfy=2*hy*(vu-vd);

// Evaluate curvature
curvature(phix,phiy,bfdelta,i,j,ij);
bfdel[ij]=bfdelta;
// Compute the updates the fluid velocity field
// The mixed density needs to be used in order to update velocity - one universal
// density field which updates the velocity field
//iden=1/rhof[ij];
iden=1/rhom[ij];
cu1[ij] = -uf[ij]*u1-vf[ij]*u2;
cu2[ij] = iden*(s11x+s12y-dt*bfdelta*phix);
cu3[ij] = iden*ex_visc*(uxx+hyy*(uu-2*uf[ij]+ud));
cu[ij] = cu1[ij] + cu2[ij] +cu3[ij];

cv1[ij] = -uf[ij]*v1-vf[ij]*v2;
cv2[ij] = iden*(s12x+s22y-dt*bfdelta*phiy);
cv3[ij] = iden*ex_vis*(vxx+hyy*(vu-2*vf[ij]+vd));
cv[ij] = cv1[ij] + cv2[ij] + cv3[ij];

crhof1[ij]=-uf[ij]*rhof1-vf[ij]*rhof2;
crhof2[ij]=-rhof[ij]*(ufx+vfy);
crhof[ij]= crhof1[ij] + crhof2[ij];

/** Calculates the x derivative of a field using either centered or one-sided
 * differencing.
 * \param[in] p the field to consider.
 * \param[in] i the horizontal gridpoint coordinate.
 * \param[in] ij the index of the gridpoint.
 * \return The computed derivative. */
inline double rmap_sim::cd_x(double *p, int i, int ij) {
    return i==0?2*(p[ij+1]-p[ij]):(i==m-1?2*(p[ij]-p[ij-1]):p[ij+1]-p[ij-1]);
}

/** Calculates the y derivative of a field using either centered or one-sided
 * differencing.
 * \param[in] p the field to consider.
 * \param[in] ij the index of the gridpoint.
 * \return The computed derivative. */
inline double rmap_sim::cd_y(double *p, int ij) {
    return ij<m?2*(p[ij+m]-p[ij]):(ij>=mn-m?2*(p[ij]-p[ij-m]):p[ij+m]-p[ij-m]);
}

/** Applies the computed updates to all of the regular fields. */
void rmap_sim::update_fields() {
    for(int ij=0; ij<mn; ij++) {
        uf[ij]+=cu1[ij]+cu2[ij]+cu3[ij];
        vf[ij]+=cv1[ij]+cv2[ij]+cv3[ij];
        rhof[ij]+=crhof[ij];

        if(c[ij]<4) {
            X[ij]+=cX[ij];
            Y[ij]+=cY[ij];
        }
    }
}
/** Calculates the ENO derivative using a sequence of values at four gridpoints.
 * \param[in] (p0,p1,p2,p3) the sequence of values to use.
 * \return The computed derivative. */
inline double rmap_sim::eno2(double p0,double p1,double p2,double p3) {
    return abs(p0-2*p1+p2)>abs(p1-2*p2+p3)?3*pl-4*p2+p3:p0-p2;
}

/** Calculates the ENO derivative of a given field at a given gridpoint.
 * \param[in] p the field to consider.
 * \param[in] ij the gridpoint index to consider.
 * \param[in] d the gridpoint displacement to consider.
 * \return The computed derivative. */
inline double rmap_sim::eno2(double *p,int ij,int d) {
    return eno2(p[ij-d],p[ij],p[ij+d],p[ij+2*d]);
}

/** Calculates the x derivative of the level set field at a given gridpoint.
 * \param[in] i the horizontal coordinate of the gridpoint.
 * \param[in] ij the gridpoint index.
 * \return The computed derivative. */
inline double rmap_sim::phi_x(int i,int ij) {
    return (i==0?phi[ij+1]-phi[ij]:i==m-1?phi[ij]-phi[ij-1]:((phi[ij+1]-phi[ij-1])*0.5))*xsp;
}

/** Calculates the y derivative of the level set field at a given gridpoint.
 * \param[in] ij the gridpoint index.
 * \return The computed derivative. */
inline double rmap_sim::phi_y(int ij) {
    return (ij<m?phi[ij+m]-phi[ij]:ij>=mn-m?phi[ij]-phi[ij-m]:((phi[ij+m]-phi[ij-m])*0.5))*ysp;
}

/** Calculates the x derivative of the psi field at a given gridpoint.
 * \param[in] ij the gridpoint index.
 * \return The computed derivative. */
inline double rmap_sim::psi_x(int i,int ij) {
    return (i==0?psi[ij+1]-psi[ij]:i==m-1?psi[ij]-psi[ij-1]:((psi[ij+1]-psi[ij-1])*0.5))*xsp;
}
/** Calculates the y derivative of the psi field at a given gridpoint. */

inline double rmap_sim::psi_y(int ij) {
    return (ij<m?psi[ij+m]-psi[ij]:ij>=mn-m?psi[ij-m]-psi[ij]:m?psi[ij+m]-psi[ij-m])*0.5)*ysp;
}

/** Saves a variety of information about the current simulation state */

void rmap_sim::write_files(const int k) {
    int ije;
    // Velocity
    output("uf",uf,k);
    output("vf",vf,k);
    output("un",un,k);
    output("erffr",erffr,k);
    output("c",c,k);
    //output("cX",cX,k);
    //output("cY",cY,k);
    //output("cu1",cu1,k);
    //output("cu2",cu2,k);
    //output("cu3",cu3,k);
    //output("cv1",cv1,k);
    //output("cv2",cv2,k);
    //output("cv3",cv3,k);
    //output("cu",cu,k);
    //output("cv",cv,k);
    //output("bfdel",bfdel,k);
    //Densities
    //output("rhof",rhof,k);
    //output_staggered("rho",rho,k);
    //output("rhom",rhom,k);
    //output("crhof",crhof,k);
    //output("crhof1",crhof1,k);
    //output("crhof2",crhof2,k);
    //output_staggered("Fract",Fract,k);
}
// Reference map
output("X",X,k);
output("Y",Y,k);

// Level set field
output("phi",phi,k);
output("psi",psi,k);
output("dpsi",dpsi,k);

// Staggered fields
output_staggered("sll",sll,k);
output_staggered("s12",s12,k);
output_staggered("s22",s22,k);

// Pressure
for(ije=0;ije<mne;ije++)
output_staggered("p",cu,k);

// Speed
for(ij=0;ij<mn;ij++)
output("sp",cu,k);

// Vorticity
for(ij=j=0;j<n;j++)
  for(i=0;i<m;i++,ij++)
    cu[ij]=0.5*(-xsp*cd_y(c[ij]<4?u:uf,i) + ysp*cd_x(c[ij]<4?v:vf,i,i));
output("vor",cu,k);

// Tracers
output_tracers(k);
// Vector field
output_vector_field(k,2*dx,0.35);

// Output the rotor mark if needed
if(sim_t==rotor) fprintf(rmarkf,"%g %g\n",k,rmarkx,rmarky);
}

/** Outputs a regular 2D array to a file in a format that can be read by
Gnuplot.
* \param[in] prefix the field name to use as the filename prefix.
* \param[in] array a pointer to the array to save.
* \param[in] sn the current frame number to append to the filename. */
template<class T>
void rmap_sim::output(const char *prefix,T *array,const int sn) {
  int i,j,ij=0;

  // Assemble the output filename and open the file
  char *bufc=((char*) buf);
  sprintf(bufc,"%s/\_%s_\%d",filename,prefix,sn);
  FILE *outf=safefopen(bufc,"wb");

  // Save the header line to the file
  *buf=m;
  for(i=0;i<m;i++) buf[i+1]=ax+(i+0.5)*dx;
  fwrite(buf,sizeof(float),m+1,outf);

  // Save the array to the file, and then close the file
  for(ij=j=0;j<n;j++) {
    *buf=ay+(j+0.5)*dy;
    for(i=0;i<m;i++,ij++) buf[i+1]=array[ij];
    fwrite(buf,sizeof(float),m+1,outf);
  }
  fclose(outf);
}

/** Outputs a staggered 2D array to a file in a format that can be read by
Gnuplot.
* \param[in] prefix the field name to use as the filename prefix.
* \param[in] array a pointer to the array to save.
* \param[in] sn the current frame number to append to the filename. */
template<class T>
void rmap_sim::output_staggered(const char *prefix,T *array,const int sn) {
  int ie,je,ije=0;

  // Assemble the output filename and open the file
  char *bufc=((char*) buf);

  // ...
```c
sprintf(bufc,"%s/%s.%d",filename,prefix,sn);
FILE *outf=safe_fopen(bufc,"wb");

// Save the header line to the file
*buf=m+1;
for(ie=0;ie<me;ie++) buf[ie+1]=ax+ie*dx;
fwrite(buf,sizeof(float),me+1,outf);

// Save the array to the file, and then close the file
for(ije=je=0;je<n+1;je++) {
    *buf=ay+je*dy;
    for(ie=0;ie<me;ie++,ije++) buf[ie+1]=array[ije];
    fwrite(buf,sizeof(float),me+1,outf);
}
fclose(outf);
}

/** Computes the normal velocity at a position vector within the simulation
 * domain using bicubic interpolation, for use within the level set update.
 * \param[in] (x,y) the position vector to consider.
 * \return The normal velocity. */
double rmap_sim::velocity(double x, double y) {
    double qun;
    qun=ls.bicubic(un,x,y);
    return -qun;
}

double rmap_sim::derivx(double *f,int ij) {
    return (f[ij]-f[ij-1])*xsp;
}

double rmap_sim::derivy(double *f,int ij) {
    return (f[ij]-f[ij-m])*ysp;
}

double rmap_sim::derivxx(double *f,int ij) {
    return (f[ij+1]-2*f[ij]+f[ij-1])*xsp*xsp;
}

double rmap_sim::derivyy(double *f,int ij,int mm) {
    return (f[ij+mm]-2*f[ij]+f[ij-mm])*ysp*ysp;
}

double rmap_sim::derivxy(double *f,int ij,int mm) {
    return (f[ij+mm+1]-f[ij+1-mm]-f[ij-1+mm]+f[ij-mm-1])*xsp*ysp*0.25;
```
/** Computes the curvature of the level set function at given gridpoint, 
* multiplied a smoothed delta function.
* \param[out] (phix,phiy) the gradient of the level set function.
* \param[out] bfdelta the curvature multiplied by a smoothed delta function.
* \param[in] (i,j) the gridpoint coordinates.
* \param[in] ij the gridpoint index. */
inline void rmap_sim::curvature(double &phix,double &phiy,double &bfdelta,
int i,int j,int ij) {
    const double pi=3.1415926535897932384626433832795,kappa=0.0; // 0.05;
    double phi=phi[ij];
    double phil=i==0?2*phi[ij+1]:phi[ij-1],phir=i==m-1?2*phi[ij-1]:phi[ij+1],
    phid=j==0?2*phi[ij+m]:phi[ij-m],phiu=j==n-1?2*phi[ij-m]:phi[ij+m];
    // Compute the gradient of the level set field
    phix=(phir-phil)*0.5*xsp;phiy=(phiu-phid)*0.5*ysp;
    // If modulus of the level set value is smaller than the smoothing
    // distance, then compute the curvature
    if (abs(phi)<eps) {
        double phidl=j==0?(i==0?2*phi[ij+m+1]:2*phi[ij+1]-phi[ij+m-1])
            *(i==0?2*phi[ij+m+1]:2*phi[ij+1]-phi[ij+m-1]),
            phidr=j==0?(i==m-1?2*phi[ij+m-1]:2*phi[ij+1]-phi[ij+m+1])
            *(i==m-1?2*phi[ij+m-1]:2*phi[ij+1]-phi[ij+m+1]),
            phiul=j==n-1?(i==0?2*phi[ij+m+1]:2*phi[ij-1]+phi[ij+m-1])
            *(i==0?2*phi[ij+m+1]:2*phi[ij-1]+phi[ij+m-1]),
            phiur=j==n-1?(i==m-1?2*phi[ij+m-1]:2*phi[ij+1]-phi[ij+m+1])
            *(i==m-1?2*phi[ij+m-1]:2*phi[ij+1]-phi[ij+m+1]);
        double phixx=(phir-2*phi[ij]+phil)*xsp*xsp,
            phixy=(phiu-phi[ij+m+1]-phi[ij-1]+phi[ij+m-1])*xsp*ysp,
            phiyy=(phiu-2*phi[ij]+phid)*ysp*ysp,
            modphi=sqrt(phixx*phixx+phiy*phiy);
        if(modphi>1e-8) {
            modphi=1/modphi;
            bfdelta=0.5*epsinv*(1+cos(pi*epsinv*phih))*(kappa*(phixx*phiy*phiy+phiy*phix-2*phixy*phi+x*phiy)*modphi*modphi);
            return;
        }
    }
    bfdelta=0;
}
Appendix D

Rmaptest.cc source code

This file represents the “main” file which is called in order to run the different simulations such as rotor, stick, circle.

```c
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sys/types.h>
#include <sys/stat.h>
using namespace std;

#define LINEAR_EXTRAPOLATION
#include "rmap_sim.hh"

int main(int argc,char **argv) {
    const int ntrace=1000;
    sim_type sim_t_;
    const char *fn;

    // Check for a single argument
    if(argc!=2) {
        puts("Syntax: ./rmap_test {circle|stick|rotor}\n",stderr);
        return 1;
    }

    // Parse the argument, giving an error if it is not one of the
    // recognized simulation types
    if(strcmp(argv[1],"circle")==0) {sim_t_=circle;fn="op_circle";}
    else if(strcmp(argv[1],"stick")==0) {sim_t_=stick;fn="op_stick";}
    else if(strcmp(argv[1],"rotor")==0) {sim_t_=rotor;fn="op_rotor";}
    else {
        fprintf(stderr,"Simulation type '%s' not understood\n",argv[1]);
    }

    // Other code...
```
return 1;
}

// Create a directory for output
mkdir(fn,S_IRWXU|S_IRWXG|S_IROTH|S_IXOTH);

// Carry out the simulation
if(sim_t==circle) {
    rmap_sim cs(sim_t,128,128,-2,2,-2,2,0.8,ntrace,fn);
    cs.solve(0,5,250);
} else if(sim_t==stick) {
    rmap_sim cs(sim_t,128,128,-2,2,-2,2,0.8,ntrace,fn);
    cs.solve(0,25,250);
} else {
    rmap_sim cs(sim_t,128,128,-2,2,-2,2,0.5,ntrace,fn);
    cs.solve(0,250,250);
}
Appendix E

Rmapsimtypes.cc source code

This file represents defines the different types of simulations. The initial settings and definitions of the level-set functions, the initial conditions, as well as deals with the boundary conditions.

```cpp
#include "rmap_sim.hh"

// Size parameters
const double prop_l=1.1;
const double prop_12=0.0;
const double prop_w=0.4;
const double prop_b=0.8;

/** Initializes the simulation fields and level set boundary. */
void rmap_sim::init_fields() {
    int i,j,ij; double x,y,xx,yy,s,xx2,yy2;
    for(ij=j=0;j<n;j++)
        for(i=0;i<m;i++,ij++) {

            // Calculate position
            x=ax+dx*(0.5+i);
            y=ay+dy*(0.5+j);
            if(sim_t!=rotor) x+=-0.5;

            // Initialize the level set function
            switch(sim_t) {
                case circle:
                    x*=0.8;x+=0.5*y;
                    phi[ij]=-0.8+sqrt(x*x+y*y);
                    break;
                case rotor:
                    xx=abs(x);yy=abs(y);
                    break;
            }
        }
}
```
if(xx<prop_b&&yy<prop_b) {
    xx-=prop_b;
    yy-=prop_b;
    phi[i][j]=prop_b-prop_w-sqrt(xx*xx+yy*yy);
} else {
    if(xx<yy) {s=xx;xx=yy;yy=s;}
    phi[i][j]=xx<prop_l?yy-prop_w:sqrt((xx-prop_l)*(xx-prop_l)+yy*yy)-prop_w;
}
break;

case stick:
    yy=abs(y)-prop_l;
    yy2=abs(y)-prop_12;
    if(yy<0) yy=0;
    if(yy2<0) yy2=0;
    xx=x;
    xx2=x+1.5;
    //yy=y; //turn it into a circle
    //xx=abs(x)+2;
    phi[i][j]=sqrt(xx*xx+yy*yy)-prop_w;
    phi2[i][j]=sqrt(xx2*xx2+yy2*yy2)-prop_w;
}

// Set the reference map at solid1 gridpoints
if(phi[i][j]<0) {
    X[i][j]=x;
    Y[i][j]=y;
} else {
    X[i][j]=Y[i][j]=0;
}

// Set the reference map at solid2 gridpoints
if(phi2[i][j]<0) {
    X2[i][j]=x;
    Y2[i][j]=y;
} else {
    X2[i][j]=Y2[i][j]=0;
}

// Set the velocities and fluid density
u[i][j]=0;

v[i][j]=0;

uf[i][j]=0;

vf[i][j]=0;

rhof[i][j]=1;

rhos[i][j]=1;
/** Computes the ghost fluid values at a given gridpoint, for the circle test case.
 * \param[in] (i,j) the coordinates of the gridpoint to consider, which may lie
 * out of bounds.
 * \param[in] ij the index of the gridpoint to consider, which may lie out of
 * bounds.
 * \param[out] (ud,vd,rhod) the ghost values of velocity and density. */

void rmap_sim::ghost_fluid_circle(int i, int j, int ij, double &ud, double &vd,
   double &rhod)
{
  if(i==-1) {
    ud=vd=0;
    rhod=1;
  } else if(i==m) {
    ud=vd=0;
    if(j==-1) {
      rhod=4*rhof[ij+m-1]-2*(rhof[ij+2*m-1]+rhof[ij+m-2])+rhof[ij+2*(m-1)];
    } else if (j==n) {
      rhod=4*rhof[ij-m-1]-2*(rhof[ij-2*m-1]+rhof[ij-m-2])+rhof[ij-2*(m+1)];
    } else {
      rhod=2*rhof[ij-1]-rhof[ij-2];
    }
  } else {
    if(j==-1) {
      ud=vd=0;
      rhod=2*rhof[ij+m]-rhof[ij+2*m];
    } else if (j==n) {
      ud=vd=0;
      rhod=2*rhof[ij-m]-rhof[ij-2*m];
    } else {
      ud=uf[ij];vd=vf[ij];rhod=rhof[ij];
    }
  }
}

/** Computes the ghost fluid values at a given gridpoint, for the stick test case.
 * \param[in] (i,j) the coordinates of the gridpoint to consider, which may lie
 * out of bounds.
 */
* \param[in] ij the index of the gridpoint to consider, which may lie out of bounds.
* \param[out] (ud,vd,rhod) the ghost values of velocity and density. */

```cpp
void rmap_sim::ghost_fluid_stick(int i, int j, int ij, double &ud, double &vd, double &rhod) {
    const double hvel=0.12;
    if(i==-i) {
        ud=hvel;
        vd=0;
        rhod=1;
    } else if(i==m) {
        ud=hvel;
        vd=0;
        if(j==-i) {
            rhod=4*rhof[ij+m-1]-2*(rhof[ij+2*m-1]+rhof[ij+m-2])+rhof[ij+2*(m-1)];
        } else if (j==n) {
            rhod=4*rhof[ij-m-1]-2*(rhof[ij-2*m-1]+rhof[ij-m-2])+rhof[ij-2*(m+1)];
        } else {
            rhod=2*rhof[ij-1]-rhof[ij-2];
        }
    } else {
        if(j==-1) {
            ud=2*uf[ij+m]-uf[ij+2*m];
            vd=0;
            rhod=2*rhof[ij+m]-rhof[ij+2*m];
        } else if (j==n) {
            ud=2*uf[ij-m]-uf[ij-2*m];
            vd=0;
            rhod=2*rhof[ij-m]-rhof[ij-2*m];
        } else {
            ud=uf[ij]; vd=vf[ij]; rhod=rhof[ij];
        }
    }
}
```

/** Computes the ghost fluid values at a given gridpoint, for the rotor test case.
* \param[in] (i,j) the coordinates of the gridpoint to consider, which may lie out of bounds.
* \param[in] ij the index of the gridpoint to consider, which may lie out of bounds.
* \param[out] (ud,vd,rhod) the ghost values of velocity and density. */

```cpp
void rmap_sim::ghost_fluid_rotor(int i, int j, int ij, double &ud, double &vd,
                                  double &rhod) {
    const double hvel=0.12;
    if(i==-i) {
        ud=hvel;
        vd=0;
        rhod=1;
    } else if(i==m) {
        ud=hvel;
        vd=0;
        if(j==-i) {
            rhod=4*rhof[ij+m-1]-2*(rhof[ij+2*m-1]+rhof[ij+m-2])+rhof[ij+2*(m-1)];
        } else if (j==n) {
            rhod=4*rhof[ij-m-1]-2*(rhof[ij-2*m-1]+rhof[ij-m-2])+rhof[ij-2*(m+1)];
        } else {
            rhod=2*rhof[ij-1]-rhof[ij-2];
        }
    } else {
        if(j==-1) {
            ud=2*uf[ij+m]-uf[ij+2*m];
            vd=0;
            rhod=2*rhof[ij+m]-rhof[ij+2*m];
        } else if (j==n) {
            ud=2*uf[ij-m]-uf[ij-2*m];
            vd=0;
            rhod=2*rhof[ij-m]-rhof[ij-2*m];
        } else {
            ud=uf[ij]; vd=vf[ij]; rhod=rhof[ij];
        }
    }
}
```
double &rhod) {
    const double spd=0.2; //0.2
    if(i==-1) {
        if(j<n/2) {
            ud=spd;
            vd=0;
            rhod=1;
        } else if(j==n) {
            ud=0;
            rhod=4*rhof[ij-m+1]-2*(rhof[ij-2*m+1]+rhof[ij-m+2])+rhof[ij-2*m+2];
        } else {
            ud=0;
            vd=2*vf[ij+1]-vf[ij+2];
            rhod=2*rhof[ij+1]-rhof[ij+2];
        }
    } else if(i==m) {
        if(j==-1) {
            ud=0;
            rhod=4*rhof[ij+m-1]-2*(rhof[ij+2*m-1]+rhof[ij+m-2])+rhof[ij+2*(m-1)];
        } else if (j==n) {
            if(i>=m/2) {
                ud=0;
                vd=spd;
                rhod=2*rhof[ij-m]-rhof[ij-2*m];
            } else {
                ud=2*uf[ij-m]-uf[ij+2*m];
                vd=0;
                rhod=2*rhof[ij+m]-rhof[ij+2*m];
            }
        } else {
            ud=uf[ij];
            vd=vf[ij];
            rhod=rhof[ij];
        }
    } else {
        if(j==-1) {
            ud=2*uf[ij+m]-uf[ij+2*m];
            vd=0;
            rhod=2*rhof[ij+m]-rhof[ij+2*m];
        } else if (j==n) {
            if(i>=m/2) {
                ud=0;
                vd=spd;
                rhod=2*rhof[ij-m]-rhof[ij-2*m];
            } else {
                ud=2*uf[ij-m]-uf[ij-2*m];
                vd=0;
                rhod=2*rhof[ij-m]-rhof[ij-2*m];
            }
        } else {
            ud=uf[ij];
            vd=vf[ij];
            rhod=rhof[ij];
        }
    }
}
/** Anchors several points in the stick simulation to have zero velocity *
* and fixed reference map values. */
void rmap_sim::anchor_stick() {
    int i,j,ij;
    double x,y;

    for(ij=j=0;j<n;j++) {
        y=ay+dy*(0.5+j);

        for(i=0;i<m;i++,ij++) {
            x=ax+dx*(0.5+i)-0.5;
            if((x*x+(y+prop-l)*(y+prop-l)<0.25*0.25)) {
                u[ij]=0;
                v[ij]=0;
                uf[ij]=0;
                vf[ij]=0;
                X[ij]=x;
                Y[ij]=y;
            }
        }
    }
}

/** Anchors the central points in the rotor axis in the rotor simulation. */
void rmap_sim::anchor_rotor() {
    int i,j,ij;
    double x,y,sxv=0,syu=0,sxx=0,syy=0,sXy=0,sYx=0,sXx=0,sYy=0;

    // Compute the angular velocity and best fit linear map for the reference map
    for(ij=j=0;j<n;j++) {
        y=ay+dy*(0.5+j);
        for(i=0;i<m;i++,ij++) {
            x=ax+dx*(0.5+i);
            if(x*x+y*y<0.16) {
                sxv+=x*vf[ij];
                syu+=y*uf[ij];
                sxx+=x*x;
                syy+=y*y;
                sXy+=X[ij]*y;
                sYx+=Y[ij]*x;
                sXx+=X[ij]*x;
                sYy+=Y[ij]*y;
            }
        }
    }
}
double omega=(sXv-sYu)/(sXx+sYy);
double theta=\text{argument}(sXx+sYy, sYx-sXy), cth=\cos(\theta), sth=\sin(\theta);

// Fix the velocity field and reference map based on the computed values
for(ij=j=0; j<n; j++) {
  y=ay+dy*(0.5+j);
  for(i=0; i<m; i++, ij++) {
    x=ax+dx*(0.5+i);
    if(x*x+y*y<0.09) {
      u[ij]=-omega*y;
      v[ij]=omega*x;
      uf[ij]=-omega*y;
      vf[ij]=omega*x;
      X[ij]=cth*x-sth*y;
      Y[ij]=sth*x+cth*y;
    }
  }
}
}
Bibliography


