PntWrks3D user Manual

Adam Yehudi Ghoneim

Department of Mechanical Engineering
University of Manitoba, Winnipeg, Manitoba R3T-5V6, Canada

19-DEC-2022

1 What is PntWrks3D?

PntWrks3D is a numerical simulation solver written in modern fortran and is dedicated to strong-form meshfree solution of partial differential equations encountered in multiphase-flow and phase transformation. This is done by discretizing an arbitrary domain geometry using fixed or moving points without connectivity information. The meshfree points are used for interpolation but can also be regarded as particles if assigned a mass.

Modern Fortran is chosen as the language of choice to write PntWrks3D because it was found to be the most efficient language that I was able to write mathematical codes in. It is simple, very fast and has many intrinsic mathematical functions that are built-in already in the language itself eliminating the need for depending on third party libraries. Earlier versions of the code were actually written in C++. Unfortunately, I was never able to make it as fast and efficient as code written in Fortran. This is most likely due to my own limited knowlege and mastery of the countless techniques to produce efficient code in C++. Afterall, I consider myself a mathematician first and a programmer second. After a number of years, C++ proved to be just too cumbersome for mathematical and numerical coding that I found myself spending a large amount of time dealing with programming paradigms and performance related issues in C++ rather than focusing on the mathematical problem itself that I am trying to solve.

The purpose of this document is to present the mathematical background and implmentation of the code library PntWrks3D. Numerical simulations and analysis data obtained using this code will also be presented and discussed in this document.

2 What can PntWrks3D do?

PntWrks3D can currently compute the strong-form meshfree solution of the following in 2 or 3 spatial dimensions and 1 temporal dimension (hence the name 4D):

- interpolation of scattered data points to approximate a function and its derivatives
- implicit surface construction of scattered data points
- solving general transport equations such as poisson, laplace, parabolic, advection-diffusion, reaction-diffusion, and equations
- solving the level-set equation
- solving the Cahn-Hilliard phase-field equation
- solving the Allen-Cahn phase-field equation
- solving the single-phase incompressible Navier-Stokes equations
- solving the two-phase incompressible Navier-Stokes equations
- solving transient elasticity and deformation equations

This is achieved via a number of meshfree interpolation methods:

- Radial basis functions
- Moving least squares
- Weighted least squares
- Least squares
- Normalized smoothed particle weighting functions
- Generalized finite difference

When the particles are fixed in space, propagating interfaces are solved using the following methods:

- volume-of-fluid
- level set
- Cahn-Hilliard or Allen-Cahn phase field
- meshfree interface tracking

3 Mathematical Theory of Interpolation Methods

In a nutshell, the basic procedure for the strong-form meshfree solution of a partial differential equation (PDE) is to approximate the field variable u_i at point $\mathbf{x}_i$ using the values u_j at neighbouring points $\mathbf{x}_j \in \{\mathbf{x}\}_{nbrs}$ such that:

$$\boxed{\hat{u}_i \approx \sum_{j=1}^{n} u_j N_{ij}} \tag{1}$$

where N_{ij} is an interpolation function that provides a weight or degree of influence from neighbouring point $\mathbf{x}_j$ on point $\mathbf{x}_i$, and n is the total number of neighbouring points. Applying the same methodology to approximate the first and second order derivative of u_i at $\mathbf{x}_i$, one can approximate the first order derivative as:

$$\nabla \hat{u}_i \approx \sum_{j=1}^{n} u_j \nabla N_{ij} \tag{2}$$

and the second order derivatives is approximated as:

$$\nabla^2 \hat{u}_i \approx \sum_{j=1}^{n} u_j \nabla^2 N_{ij} \tag{3}$$

Note that if the field variable u_j is constant then the approximation in 2 will yield a non-zero answer. A better approximation of the first order derivative is:

$$\boxed{\nabla \hat{u}_i \approx \sum_{j=1}^{n} \Big(u_j - u_i\Big) \nabla N_{ij}} \tag{4}$$

Directly computing the second-order derivative by directly computing the second-order derivative of the interpolation function as presented in equation 3 is prone to instabilities and is very sensitive to particle distribution. One approach is to approximate the second-order derivative by differentiating the first-order derivatives of the weighting functions twice such that:

$$\nabla^2 \hat{u}_i = \nabla \cdot \nabla \hat{u}_i \approx \sum_{j=1}^{n} \Big(\nabla u_j - \nabla u_i\Big) \cdot \nabla N_{ij} \tag{5}$$

A better approach uses a Taylor series expansion in a similar manner to finite differences to achieve the approximation:

$$\boxed{\nabla^2 \hat{u}_i \approx 2 \sum_{j=1}^{n} \frac{\Big(u_j - u_i\Big)}{|\mathbf{r}_{ji}|^2} \mathbf{r}_{ji} \cdot \nabla N_{ij}} \tag{6}$$

where $\mathbf{r}_{ji} = \mathbf{x}_j - \mathbf{x}_i$ is the Unit vector between interacting points $\mathbf{x}_j$ and $\mathbf{x}_i$.

It is interesting to note that one can actually approximate the first and second order derivatives using only zero-order interpolants such that:

$$\boxed{\nabla \hat{u}_i \approx \sum_{j=1}^{n} \mathbf{r}_{ij} \Big(u_j - u_i \Big) N_{ij}} \tag{7}$$

and

$$\boxed{\nabla^2 \hat{u}_i \approx \sum_{j=1}^{n} \frac{1}{|\mathbf{r}_{ji}|^2} \Big(u_j - u_i \Big) N_{ij}} \tag{8}$$

These three equations used to approximate a field variable u_i and its derivatives form the backbone of PntWrks3D. Once the meshfree derivatives of field variables can be calculated, it can be used directly for solving a wide range of PDE's. However, obviously to achieve this, we must first somehow evaluate the interpolation functions N_{ij} and at the very least their first-order derivatives. In the following sections I present the mathematical derivations of the various methods used in PntWrks3D.

3.1 Smoothed Kernel Approximation

The SPH is based on approximating a field variable u_i at point $\mathbf{x}_i$ by:

$$u_i = \int_{\Omega} u_j \delta_{ij} \Delta V \tag{9}$$

where j is a neighbouring point, Ω is the volume of the domain containing $\mathbf{x}$ and δ_{ij} is the Dirac delta function defined by:

$$\delta_{ij} = \begin{cases} 1, & \mathbf{x}_i = \mathbf{x}_j \\ 0, & \mathbf{x}_i \neq \mathbf{x}_j \end{cases} \tag{10}$$

To establish a meaningful discrete numerical model, the delta function δ_{ij} is replaced by a smooth kernel function $W(\mathbf{x}_i - \mathbf{x}_j, h) = w_{ij}$ where h is a kernel smoothing length defining the size of the support area. The function u_i can then be approximated by $\hat{u}_i$ by:

$$\hat{u}_i \approx \int_{\Omega} u_j w_{ij} \Delta V \tag{11}$$

Equation 11 can be re-written in discrete form for point $\mathbf{x}_i$ in terms of the set of neighbouring particles $\{\mathbf{x}\}_{nbrs}$ such that:

$$\hat{u}_i \approx \sum_{j=1}^{n} u_j w_{ij} \Delta V_j = \sum_{j=1}^{n} u_j N_{ij} \tag{12}$$

where n is the total number of points in the kernel containing $\{\mathbf{x}\}_{nbrs}$ and the term $w_{ij}\Delta V_j$ forms an interpolant N_{ij} that weighs the influence of a neighbor point $\mathbf{x}_j$ on the solution at $\mathbf{x}_i$ such that its influence decreases as it gets farther from $\mathbf{x}_i$. Indeed, it could be argued that this is basically the approximation approach of all strong-form meshfree methods that were developed after the SPH where they mainly differ in the manner of constructing the interpolant N_{ij}. Equation 12 can be re-expressed as:

$$\hat{u}_i \approx \sum_{j=1}^{n} u_j w_{ij} \Delta V_j = \sum_{j=1}^{n} u_j w_{ij} \frac{1}{\rho_j} (\rho_j \Delta V_j) \tag{13}$$

or alternatively:

$$\hat{u}_i \approx \sum_{j=1}^{n} u_j w_{ij} \frac{m_j}{\rho_j} \tag{14}$$

where ρ_j and m_j is the density and mass of neighbouring point j, respectively. Notice that if a density and mass is assigned to a point, it can now be regarded as a particle.

To choose a suitable kernel function w_{ij} within the context of SPH, Monaghan proposed three main conditions. The first is the so called normalization (or Unity) condition defined as:

$$\int_{\Omega} w_{ij} \Delta V = 1 \tag{15}$$

The second condition is the so called Delta function property defined as:

$$\lim_{h \to 0} w_{ij} \Delta V = \delta_{ij} \tag{16}$$

This means that as the kernel smoothing length h decreases, the delta property in equation 10 is recovered. The third condition is the compactness condition defined as:

$$w_{ij} = 0 \quad \forall \quad |\mathbf{x}_i - \mathbf{x}_j| > h \tag{17}$$

A popular kernel function satisfying these conditions is the Gaussian kernel defined by:

$$w_{ij} = \frac{1}{\pi^{(d/2)} h^d} e^{-R^2} \tag{18}$$

where d is the spatial dimension of the problem $d = \{1, 2, 3\}$ and $R = \frac{|\mathbf{x}_i - \mathbf{x}_j|}{h} = \frac{r_{ij}}{h}$. Another kernel function is the cubic spline kernel function expressed as:

$$w_{ij} = C_h \begin{cases} (2-R)^3 - 4(1-R)^3 & if \quad 0 \leq R < 1 \\ (2-R)^3 & if \quad 1 \leq R < 2 \\ 0, & if \quad q \geq 2 \end{cases} \tag{19}$$

where C_h is $1/(6h)$ in one dimension, $15/(14\pi h^2)$ in two dimensions and $1/(4\pi h^3)$ in three dimensions. Wendland kernel functions have also been used in SPH. One type is the so-called quintic Wendland kernel function expressed as:

$$w_{ij} = C_h \begin{cases} (2-R)^4 (2R+1) & if \quad 0 \leq R \leq 2 \\ 0 & if \quad R > 2 \end{cases}$$

(20)

where C_h is $7/(4\pi h^2)$ in 2D and C_h is $21/(16\pi h^3)$ in 3D.

The main difficulty in using the kernel functions in 18, 19 and 20 directly into the approximation in equation 12 is that it fails even the zero-order consistency (reproducibility) condition where $\sum_{j=1}^{n} w_{ij}\Delta V_j \neq 1$ at the domain boundaries due to the kernel being only partially contained within the domain. In other words, the kernel functions used in classical SPH will not satisfy the partition of Unity (POu) condition necessary for consistency. Exhibiting the consistency property is a necessary ingredient to achieve convergence of a numerical solution. An easy approach to resolve this issue is to re-assign the weights to points within the kernel such that the POu property is recovered. This can be done by through normalization:

$$\hat{u}_i \approx \frac{1}{D}\sum_{j=1}^{n} u_j w_{ij}\Delta V_j \tag{21}$$

where $D = \sum_{j=1}^{n} w_{ij}\Delta V_j$ and

$$\frac{1}{D}\sum_{j=1}^{n} w_{ij}\Delta V_j = \sum_{j=1}^{n} \frac{w_{ij}\Delta V_j}{\sum_{j=1}^{n} w_{ij}\Delta V_j} = 1 \tag{22}$$

The interpolation function between point i and neighbour point j can then be expressed as:

$$\boxed{N_{ij} = \frac{w_{ij}}{\sum_{j=1}^{n} w_{ij}}} \tag{23}$$

similary, the first and second order derivatives can be approximated as:

$$\boxed{\nabla N_{ij} = \frac{\nabla w_{ij}}{\sum_{j=1}^{n} w_{ij}}} \tag{24}$$

$$\boxed{\nabla^2 N_{ij} = \frac{\nabla^2 w_{ij}}{\sum_{j=1}^{n} w_{ij}}} \tag{25}$$

It should be noted that this approach of normalizing interpolants coincides with Shepard's interpolation which is itself considered a special case of MLS approximation. However, Shepard's interpolation is only zero-order continuous. To Achieve high order of consistency we next look at more sophisticated approaches to construct the interpolants. It should also be noted that in PntWrks3D, we actually don't use equation 25 to approximate second order derivatives directly but rather use equation 8 stated earlier.

3.2 Radial Basis Function Interpolation

To construct the RBF interpolants, the Euclidean distance r_{ij} between a point $\mathbf{x}_i$ to a neighbouring point $\mathbf{x}_j$ is computed such that:

$$r_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| \tag{26}$$

Common RBF functions include multiquadrics, inverse multiquadrics, Gaussian exponential, Thin-plate spline and Logarithmic RBF. The Gaussian exponential RBF is expressed as:

$$R_{ij} = exp(-\gamma(\frac{r_{ij}}{h})^2) \tag{27}$$

where h is the size of the kernel radius, γ is a shape parameter that is often problem dependent. Notice that an attractive property of this function is that regardless of the order of differentiation, it is continuously differentiable. A smaller shape parameter yields a wider, or flatter, weight function which will therefore alter the constructed RBF interpolant. It has been reported in the literature that the optimum shape parameter for Gaussian exponential basis yielding best accuracy is in the range of $\gamma = 0.003 - 0.03$.

For our point $\mathbf{x}_i$, the field variable u_i can be approximated by RBF such that:

$$\hat{u_i} = \sum_{j=1}^{n} R_{ij}a_j + \sum_{k=1}^{m} P_{ik}b_k = \begin{bmatrix} R_i & P_i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \tag{28}$$

where R_{ij} is a RBF evaluated at the point $\mathbf{x}_i$, P_{ik} is a monomial evaluated at $\mathbf{x}_i$, n is the total number of neighbouring points, m is the number of polynomial basis functions while a_j and b_k are unknowns to be determined. For a second order polynomial basis function:

$$P = \begin{bmatrix} 1, & x_i, & y_i, & z_i, & x_i^2, & y_i^2, & z_i^2, & x_iy_i, & x_iz_i, & y_iz_i \end{bmatrix} \tag{29}$$

To satisfy equation 28 at all neighbouring points, we express it in matrix form:

$$\mathbf{u} = \begin{bmatrix} \mathbf{R} & \mathbf{P} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \tag{30}$$

where

$$\mathbf{u} = \begin{bmatrix} u_1, & u_2, & u_3, & ...., & u_n \end{bmatrix}^T \tag{31}$$

$\mathbf{R}$ is an n x n matrix, and $\mathbf{P}$ is an n x m matrix. Clearly there are $n + m$ unknowns and only n equations. Therefore, we require to impose the constraint:

$$\mathbf{u} = \begin{bmatrix} \mathbf{R} & \mathbf{P} \\ \mathbf{P} & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \mathbf{A} \begin{bmatrix} a \\ b \end{bmatrix} \tag{32}$$

To determine the unknowns $[a]$ and $[b]$ we can re-express this equation to be

$$\begin{bmatrix} a \\ b \end{bmatrix} = \mathbf{A}^{-1}\mathbf{u} \tag{33}$$

which can then be substituted back into equation 28 to yield:

$$\hat{u_i} = \begin{bmatrix} R_i & P_i \end{bmatrix} \mathbf{A}^{-1}\mathbf{u} = \mathbf{N}\mathbf{u} \tag{34}$$

where $\mathbf{N}$ are the set of interpolation functions $\mathbf{N} = \begin{bmatrix} N_1, N_2, N_3, ..., N_n \end{bmatrix}$ computed by:

$$\boxed{\mathbf{N} = \begin{bmatrix} R_i & P_i \end{bmatrix} \mathbf{A}^{-1}} \tag{35}$$

First and second order derivatives of the constructed RBF interpolants can be computed by:

$$\boxed{\frac{\partial \mathbf{N}}{\partial \delta} = \begin{bmatrix} \frac{\partial R_i}{\partial \delta} & \frac{\partial P_i}{\partial \delta} \end{bmatrix} \mathbf{A}^{-1}} \tag{36}$$

where $\frac{\partial}{\partial \delta}$ is a partial derivative with respect to $\delta = \{x, y, z, xx, yy, zz, xy, xz, yz\}$.

Properties of the constructed interpolants include the satisfaction of the partition of Unity property:

$$\sum_{j=1}^{n} N_{ij} = 1 \tag{37}$$

In addition to the Kronecker delta property:

$$N_{ij} = \begin{Bmatrix} 1 & for \quad j = i \\ 0 & for \quad j \neq i \end{Bmatrix} \tag{38}$$

and the reproducing property:

$$\sum_{j=1}^{n} u N_{ij} = u \tag{39}$$

3.3 Moving Least Squares Approximantion

Consider a point $\mathbf{x}_i$. A field variable solution u_i can be approximated by using a polynomial such that:

$$\hat{u_i} = \sum_{k=1}^{m} p_{ik} a_k = \mathbf{p}^T \mathbf{a} \tag{40}$$

where m is the number of monomials, a are unknown coefficients, and for a second order polynomial p is expressed as:

$$p^T = [1, x, y, z, x^2, y^2, z^2, xy, xz, yz] \tag{41}$$

Let us introduce a weighting function and minimize the weighted residual such that:

$$J = \sum_{i=1}^{n} w(\hat{u} - u)^2 \tag{42}$$

where n is the neighbouring points, w is a weighting function and $\hat{u}$ is an approximate value of u. This equation can be re-expressed as:

$$J = \sum_{j=1}^{n} w\Big(\sum_{k=1}^{m} p_{ik} a_k - u \Big)^2 \tag{43}$$

which can be rewriten in the form:

$$J = \Big(\mathbf{Pa} - \mathbf{u} \Big)^T \mathbf{W} \Big(\mathbf{Pa} - \mathbf{u} \Big) \tag{44}$$

we minimize J by differentiating it and forcing it to zero such that:

$$\frac{\partial J}{\partial \mathbf{a}} = \Big(\mathbf{P}^T\mathbf{W}\mathbf{P}\Big)\mathbf{a} - \Big(\mathbf{P}^T\mathbf{W}\Big)\mathbf{u} = 0 \tag{45}$$

where

$$\mathbf{u}^T = [u_1, u_2, u_3, ..., u_n] \tag{46}$$

and the coefficients $\mathbf{a}$ are constantly changing, or moving and, hence the name "moving least squares".

In the MLS, a distance-dependent weighting function is used such that:

$$w(r_{ij}) = \begin{cases} w_g(r_{ij}) & r_{ij} \leq h \\ 0 & r_{ij} > h \end{cases} \tag{47}$$

where h is the kernel radius and r_{ij} is the Eucledian distance between the point $\mathbf{x}_i$ and neighbouring point $\mathbf{x}_j$ such that

$$r_{ij} = |\mathbf{x}_i - \mathbf{x}_j| \tag{48}$$

common weighting functions is the Gaussian weighting function of the exponential type

$$w_g(r_{ij}) = \frac{e^{-(r_{ij}/\alpha h)^2} - e^{-1/\alpha^2}}{1 - e^{-1/\alpha^2}} \tag{49}$$

where α is a shape parameter.

This results in a weighting function matrix such that:

$$\mathbf{W}(\mathbf{x}) = \begin{bmatrix} w_1(\mathbf{x}_1) & w_1(\mathbf{x}_2) & w_1(\mathbf{x}_3) & ... & w_1(\mathbf{x}_n) \\ w_2(\mathbf{x}_1) & w_2(\mathbf{x}_2) & w_2(\mathbf{x}_3) & ... & w_2(\mathbf{x}_n) \\ w_3(\mathbf{x}_1) & w_3(\mathbf{x}_2) & w_3(\mathbf{x}_3) & ... & w_3(\mathbf{x}_n) \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ w_m(\mathbf{x}_1) & w_m(\mathbf{x}_2) & w_m(\mathbf{x}_3) & ... & w_m(\mathbf{x}_n) \end{bmatrix}$$

for Weighted Least Squares this is simplified to be:

$$\mathbf{W}(\mathbf{x}) = \begin{bmatrix} w_1(\mathbf{x}_1) & 0 & 0 & ... & 0 \\ 0 & w_2(\mathbf{x}_2) & 0 & ... & 0 \\ 0 & 0 & w_3(\mathbf{x}_3) & ... & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 0 & 0 & ... & w_m(\mathbf{x}_n) \end{bmatrix}$$

if the diagonals of the weight functions matrix are set to 1, the method reduces to the standard Least squares approximation.

(50)

let

$$\mathbf{A} = \mathbf{P}^T\mathbf{W}\mathbf{P} \tag{51}$$

and

$$\mathbf{B} = \mathbf{P}^T\mathbf{W} \tag{52}$$

Then the set of unknown coefficients $\mathbf{a}$ can be obtained from

$$\mathbf{a} = \mathbf{B}\mathbf{A}^{-1}\mathbf{u} \tag{53}$$

one can then substitute it back into equation 40 to obtain $u(\mathbf{x})$ such that

$$u_i = \mathbf{p}^T\mathbf{B}\mathbf{A}^{-1}\mathbf{u} \tag{54}$$

or simply

$$u_i = \mathbf{N}\mathbf{u} \tag{55}$$

where $\mathbf{N}$ are the MLS interpolation functions defined as

$$\boxed{\mathbf{N} = \mathbf{p}^T\mathbf{B}\mathbf{A}^{-1}} \tag{56}$$

let

$$\gamma^T = \mathbf{p}^T\mathbf{A}^{-1} \tag{57}$$

$$\mathbf{A}\gamma^T = \mathbf{p} \tag{58}$$

We follow the technique proposed by Belytschko et. al. to compute the partial derivatives of the interpolants sunch that:

$$\boxed{\mathbf{N}^T_\alpha = \gamma^T_\alpha\mathbf{B} + \gamma^T\mathbf{B}_\alpha} \tag{59}$$

$$\boxed{\mathbf{N}^T_{\alpha\beta} = \gamma^T_{\alpha\beta}\mathbf{B} + \gamma^T_\alpha\mathbf{B}_\beta + \gamma^T_\beta\mathbf{B}_\alpha + \gamma^T\mathbf{B}_{\alpha\beta}} \tag{60}$$

where subscripts $\alpha = \{x, y, z\}$ and $\beta = \{x, y, z\}$ correspond to the partial derivative with respect to spatial co-ordinates x, y and z.

Due to the least squares approach, the MLS interpolation does not pass through the nodal values. This implies the interpolation condition is not fulfilled

$$N_{ij} \neq \delta_{ij} \tag{61}$$

Most and Buchar proposed that in order to satisfy

$$N_{ij} = \delta_{ij} \tag{62}$$

this can only be satisfied if

$$w_{ij} = \delta_{ij} \tag{63}$$

The weighting function value an an interpolation point $\mathbf{x}_i$ is introduced by the following regularized formulation:

$$w_{ij} = \frac{w^*_{ij}}{\sum_{j=1}^{n} w^*_{ij}} \tag{64}$$

such that:

$$w_{ij}^{*} = \frac{A^{-2} - (1+\alpha)^{-2}}{B} \tag{65}$$

where

$$A = \left(\frac{r_{ij}}{h}\right)^{2} + \alpha \tag{66}$$

and

$$B = \alpha^{-2} - (1+\alpha)^{-2} \tag{67}$$

such that n is the total number of neighbouring points in the kernel and α must be small enough to fulfill equation 63 and larger than the square root of machine precision to avoid numerical issues. A closer look into equation 65 would indicate that if the eucledian distance $r_{ij} = 0$ (meaning neighbouring point $\mathbf{x}_j$ coincides with point $\mathbf{x}_i$) equation 65 reduces to $w_{ij}^{*} = 1$ while if $r_{ij} > 0$, $w_{ij}^{*} \approx 0$. In other words, the Kronecker-delta condition is recovered to high degree of accuracy. The first and second order derivatives of the weight function can be easily computed as:

$$\frac{\partial w_{ij}}{\partial r_{ij}} = \frac{-4r_{ij}}{h^2}\frac{A^{-3}}{B} \tag{68}$$

$$\frac{\partial^2 w_{ij}}{\partial r_{ij}^2} = \frac{1}{B}\left(\frac{24r_{ij}^2}{h^4}A^{-4} - \frac{4}{D^2}A^{-3}\right) \tag{69}$$

At the influence boundary where $r_{ij} = h$, the derivatives of the weight function can be approximated as:

$$\frac{\partial w_{ij}}{\partial r_{ij}} = \frac{4\epsilon^2}{h} \tag{70}$$

$$\frac{\partial^2 w_{ij}}{\partial r_{ij}^2} = \frac{20\epsilon^2}{h^2} \tag{71}$$

3.4 Generalized Finite Differences

In contrast to meshfree approximation methods based on smoothing kernel interpolants, the Generalized Finite Difference Method (GFDM) is based on Taylor series approximation. For a neighbouring point $\mathbf{x}_j$ in the kernel to the center point $\mathbf{x}_i$, the field variable u_j can be related to u_i using Taylor series by:

$$u_j = u_i + h_{ij}\frac{\partial u_i}{\partial x} + k_{ij}\frac{\partial u_i}{\partial y} + l_{ij}\frac{\partial u_i}{\partial z} + \frac{h_{ij}^2}{2}\frac{\partial^2 u_i}{\partial x^2} + \frac{k_{ij}^2}{2}\frac{\partial^2 u_i}{\partial y^2} + \frac{l_{ij}^2}{2}\frac{\partial^2 u_i}{\partial z^2} + h_{ij}k_{ij}\frac{\partial^2 u_i}{\partial x \partial y} + h_{ij}l_{ij}\frac{\partial^2 u_i}{\partial x \partial z} + k_{ij}l_{ij}\frac{\partial^2 u_i}{\partial y \partial z} \tag{72}$$

where $h_{ij} = x_i - x_j$, $k_{ij} = y_i - y_j$, $l_{ij} = z_i - z_j$. A functional $J(u)$ is introduced such that

$$J(u) = \sum_{j=1}^{n}\Bigg(\Big(u_i - u_j + h_{ij}\frac{\partial u_i}{\partial x} + k_{ij}\frac{\partial u_i}{\partial y} + l_{ij}\frac{\partial u_i}{\partial z} + \frac{h_{ij}^2}{2}\frac{\partial^2 u_i}{\partial x^2} + \frac{k_{ij}^2}{2}\frac{\partial^2 u_i}{\partial y^2} + \frac{l_{ij}^2}{2}\frac{\partial^2 u_i}{\partial z^2} + h_{ij}k_{ij}\frac{\partial^2 u_i}{\partial x \partial y} + h_{ij}l_{ij}\frac{\partial^2 u_i}{\partial x \partial z} + k_{ij}l_{ij}\frac{\partial^2 u_i}{\partial y \partial z}\Big)w_{ij}\Bigg)^2$$

(73)

where w_{ij} is a weighting function which can be of various forms such as cubic spline, or quartic spline. A quartic spline weight function can be expressed as:

$$w(r_{ij}) = \begin{cases} 1 - 6\Big(\frac{r_{ij}}{h}\Big)^2 + 8\Big(\frac{r_{ij}}{h}\Big)^3 - 3\Big(\frac{r_{ij}}{h}\Big)^4 & r_{ij} \leq h \\ 0 & r_{ij} > h \end{cases} \tag{74}$$

where $r_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$. To minimize the functional $J(u)$ with respect to the set of derivatives:

$$\mathbf{D} = \left[\frac{\partial u_i}{\partial x}, \quad \frac{\partial u_i}{\partial y}, \quad \frac{\partial u_i}{\partial z}, \quad \frac{\partial^2 u_i}{\partial x^2}, \quad \frac{\partial^2 u_i}{\partial y^2}, \quad \frac{\partial^2 u_i}{\partial z^2}, \quad \frac{\partial^2 u_i}{\partial x \partial y}, \quad \frac{\partial^2 u_i}{\partial x \partial z}, \quad \frac{\partial^2 u_i}{\partial y \partial z}\right]^T \tag{75}$$

Let's define:

$$H_{ij} = \left[\Delta x_{ij}, \quad \Delta y_{ij}, \quad \Delta z_{ij}, \quad \frac{\Delta x_{ij}^2}{2}, \quad \frac{\Delta y_{ij}^2}{2}, \quad \frac{\Delta z_{ij}^2}{2}, \quad \Delta x_{ij}\Delta y_{ij}, \quad \Delta x_{ij}\Delta z_{ij}, \quad \Delta y_{ij}\Delta z_{ij}\right]^T \tag{76}$$

for all n neighbouring points to central point i in the kernel this becomes:

$$\mathbf{H} = \begin{bmatrix} \Delta x_{i1} & \Delta y_{i1} & \Delta z_{i1} & ... & \Delta y_{i1}\Delta z_{i1} \\ \Delta x_{i2} & \Delta y_{i2} & \Delta z_{i2} & ... & \Delta y_{i2}\Delta z_{i2} \\ \Delta x_{i3} & \Delta y_{i3} & \Delta z_{i3} & ... & \Delta y_{i3}\Delta z_{i3} \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ \Delta x_{in} & \Delta y_{in} & \Delta z_{in} & ... & \Delta y_{in}\Delta z_{in} \end{bmatrix}$$

The functional can then be rewritten as:

$$J(u) = \Big(\mathbf{HD} + \mathbf{u}_i - \mathbf{u}\Big)^T \mathbf{W}\Big(\mathbf{HD} + \mathbf{u}_i - \mathbf{u}\Big) \tag{77}$$

where

$$\mathbf{u} = [u_1, \quad u_2, \quad u_3, \quad ... \quad u_n]^T \tag{78}$$

and

$$\mathbf{u}_i = [u_i, \quad u_i, \quad u_i, \quad ... \quad u_i]^T \tag{79}$$

and

$$\mathbf{W} = \begin{bmatrix} w_1^2 & 0 & 0 & ... & 0 \\ 0 & w_2^2 & 0 & ... & 0 \\ 0 & 0 & w_3^2 & ... & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 0 & 0 & ... & w_n^2 \end{bmatrix}$$

Minimizing the functional J with respect to $\mathbf{D}$ such that

$$\frac{\partial J}{\partial \mathbf{D}} = 0 \tag{80}$$

results in a system of equations

$$\mathbf{AD} = \mathbf{b} \tag{81}$$

where

$$\mathbf{A} = \mathbf{H}^T\mathbf{W}\mathbf{H} \tag{82}$$

and

$$\mathbf{b} = \mathbf{H}^T\mathbf{W}(\mathbf{u} - \mathbf{u}_i) \tag{83}$$

we can then compute the derivatives of u_i by solving for $\mathbf{D}$ such that:

$$\mathbf{D} = \mathbf{A}^{-1}\mathbf{b} \tag{84}$$

Notice that $\mathbf{W}$, $\mathbf{H}$ and $\mathbf{A}$ are all functions of only the spatial distances between point $\mathbf{x}_i$ and its neighbours. So for a fixed set of points, one needs to calculate them only once and then calculate the inverse matrix $\mathbf{A}^{-1}$ also only once. The results are then stored in memory and reused to re-calculate $\mathbf{b}$ as the field variables u change from one time step to the next. A new result for $\mathbf{D}$ is then obtained by multiplying the stored $\mathbf{A}^{-1}$ by the newly calculated $\mathbf{b}$ to determine a new approximation of the derivatives $\mathbf{D}$ whose values are used to solve the PDE directly.

4 Time Stepping Schemes

In PntWrks3D we currently use explicit time stepping schemes. This is mainly because explicit schemes avoid the need to solve a global system of equations which is often time consuming to solve. The solution at each kernel can be done independenly from the other kernel allowing for simple parallelization. Obviously, using explicit schemes limits the size of time steps that can be used to ensure a stable solution. However, I have found that even with smaller time steps, the solver is almost always more efficient than when using implicit schemes where large systems of equations must be solved at each time step.

In the following equations, superscript t indicates the time step, Δt is the size of the time step, u is the field variable, and $f(u)$ is the PDE with respect to u.

Explicit Forward Euler

$$u^{t+1} = u^t + \Delta t f(u)^t \tag{85}$$

Adam-Bashforth Order 1

$$u^{t+1} = u^t + \Delta t f(u)^t \tag{86}$$

Adam-Bashforth Order 2

$$u^{t+1} = u^t + \Delta t\Big(\frac{3}{2}f(u)^t - \frac{1}{2}f(u)^{t-1}\Big) \tag{87}$$

Adam-Bashforth Order 3

$$u^{t+1} = u^t + \Delta t\Big(\frac{23}{12}f(u)^t - \frac{16}{12}f(u)^{t-1} + \frac{5}{12}f(u)^{t-2}\Big) \tag{88}$$

Adam-Bashforth Order 4

$$u^{t+1} = u^t + \Delta t\Big(\frac{55}{24}f(u)^t - \frac{59}{24}f(u)^{t-1} + \frac{37}{24}f(u)^{t-2} - \frac{9}{24}f(u)^{t-3}\Big) \tag{89}$$

Stability Criterion for Diffusion

$$\frac{\beta\Delta t}{(\Delta x)^2} + \frac{\beta\Delta t}{(\Delta y)^2} + \frac{\beta\Delta t}{(\Delta z)^2} \leq \frac{1}{2} \tag{90}$$

where $\beta = \frac{k}{\rho c}$, k is thermal conduction, ρ is the density and c is the thermal capacity

CFL Condition

$$\frac{V_{max}\Delta t}{\Delta x} \leq C_{max} \tag{91}$$

5 Partial Differential Equations

The General Transport Equation

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\nabla u) - \nabla \cdot (u\mathbf{V}) + R(u) \tag{92}$$

or in linear form:

$$\frac{\partial u}{\partial t} = D\nabla^2 u - \mathbf{V} \cdot \nabla u + R \tag{93}$$

where u is the field variable, D is diffusivity, $\mathbf{V}$ is advective velocity, R is generation/reaction term.

The Reaction-Diffusion Equation

$$\frac{\partial u}{\partial t} = D\nabla^2 u + R(u) \tag{94}$$

The Parabolic Equation

$$\frac{\partial u}{\partial t} = D\nabla^2 u \tag{95}$$

Laplace's Equation

$$0 = D\nabla^2 u \tag{96}$$

Poisson's Equation

$$f = D\nabla^2 u \tag{97}$$

Burger's Equation

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} - u\frac{\partial u}{\partial x} \tag{98}$$

Inviscid Burger's Equation

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} \tag{99}$$

The Level Set Equation

$$\frac{\partial \Phi}{\partial t} = b\kappa\nabla^2\Phi - V_n^{ext}|\nabla\Phi| - \mathbf{V} \cdot \nabla\Phi \tag{100}$$

where Φ is the level set function parameter, V_n^{ext} is the extended interface velocity, $\mathbf{V}$ is advective velocity, b is a diffusion constant, κ is curvature of all level set contours.

Curvature-driven flow

$$\frac{\partial \Phi}{\partial t} = M_\Phi\Big(\nabla^2\Phi + \frac{\Phi(1-\Phi^2)}{\epsilon^2}\Big) \tag{101}$$

to describe the case of no curvature-driven interface motion:

$$\frac{\partial \Phi}{\partial t} = M_\Phi\Big(\nabla^2\Phi + \frac{\Phi(1-\Phi^2)}{\epsilon^2} - |\nabla\Phi|\nabla\cdot\Big(\frac{\nabla\Phi}{|\nabla\Phi|}\Big)\Big) \tag{102}$$

The Ginzburg-Landau Level Set Equation

$$\frac{\partial \Phi}{\partial t} = \epsilon\nabla^2\Phi - V_n^{ext}|\nabla\Phi| - \mathbf{V}\cdot\nabla\Phi \tag{103}$$

The Reaction-Diffusion Level Set Equation

$$\frac{\partial \Phi}{\partial t} = \epsilon_1\nabla^2\Phi - \frac{1}{\epsilon_2}\Big(V_n^{ext}|\nabla\Phi| + \mathbf{V}\cdot\nabla\Phi\Big) \tag{104}$$

The Conservative Level Set Equation

$$\frac{\partial \Phi}{\partial t} + \mathbf{V}\cdot\nabla\Phi = \gamma\nabla\cdot\Big(\epsilon\nabla\Phi - \Phi(1-\Phi)\frac{\nabla\Phi}{|\nabla\Phi|}\Big) \tag{105}$$

$$0 \le \Phi \le 1 \tag{106}$$

The Sharp-Interface Phase Field Equation

$$\frac{\partial \Psi}{\partial t} = \epsilon\Big(\nabla^2\Psi + \frac{\Psi(1-\Psi^2)}{W^2} - |\nabla\Psi|\kappa\Big) - V_n^{ext}|\nabla\Psi| - \mathbf{V}\cdot\nabla\Psi \tag{107}$$

The Cahn-Hilliard Equation

$$\frac{\partial \Psi}{\partial t} + \mathbf{V}\cdot\nabla\Psi = D\nabla^2(\Psi^3 - \Psi - \gamma\nabla^2\Psi) \tag{108}$$

The Cahn-Hilliard Equation with Constant Mobility

$$\frac{\partial \Psi}{\partial t} + \mathbf{V}\cdot\nabla\Psi = M\nabla^2\mu(\Psi) \tag{109}$$

$$\mu(\Psi) = F'(\Psi) - \epsilon\nabla^2\Psi \tag{110}$$

$$F'(\Psi) = \Psi(\Psi^2 - 1) \tag{111}$$

The Cahn-Hilliard Equation with Variable Mobility

$$\frac{\partial \Psi}{\partial t} + \mathbf{V} \cdot \nabla \Psi = \nabla \cdot \Big(M(\Psi) \nabla \mu(\Psi) \Big) \tag{112}$$

$$\mu(\Psi) = F'(\Psi) - \epsilon \nabla^2 \Psi \tag{113}$$

$$F(\Psi) = 0.25(\Psi^2 - 1)^2, F(\Psi) = 0.25\Psi^2(\Psi - 1)^2 \tag{114}$$

$$F'(\Psi) = \Psi(\Psi^2 - 1) \tag{115}$$

$$M(\Psi) = 1 - \Psi^2, M(\Psi) = \Psi(1 - \Psi), M(\Psi) = 1 \tag{116}$$

The Cahn-Hilliard Equations for Spinodal Decomposition

$$\frac{\partial \Psi}{\partial t} + \mathbf{V} \cdot \nabla \Psi = \nabla^2 \Big(M \frac{\delta F}{\delta \Psi} \Big) \tag{117}$$

$$\frac{\delta F}{\delta \Psi} = \mu(\Psi) - K \nabla^2 \Psi \tag{118}$$

$$\mu(\Psi) = A \Big(2\Psi(1 - \Psi)^2 + 2(\Psi)^2(1 - \Psi) \Big) \tag{119}$$

where $0 <= \Psi <= 1$

The Allen-Cahn equation

$$\frac{\partial \Psi}{\partial t} + \mathbf{V} \cdot \nabla \Psi = M \big(\epsilon^2 \nabla^2 \Psi - F'(\Psi) \big) \tag{120}$$

where M is the mobility, F is the free energy density and Ψ is the nonconserved order parameter.

The Conservative Allen-Cahn equation

$$\frac{\partial \Psi}{\partial t} + \mathbf{V} \cdot \nabla \Psi = \epsilon^2 \nabla^2 \Psi - F'(\Psi) + \alpha(t) \tag{121}$$

$$F'(\Psi) = \Psi(\Psi^2 - 1) \tag{122}$$

where $\alpha(t)$ is a time dependent lagrange multipler which enforces mass conservation such that:

$$\int_\Omega \Psi(\mathbf{x}, t) d\mathbf{x} = constant \tag{123}$$

such that:

$$\alpha(t) = \frac{\int_\Omega F'(\Psi(\mathbf{x}, t)) d\mathbf{x}}{\int_\Omega d\mathbf{x}} \tag{124}$$

The Conservative Allen-Cahn equation

$$\frac{\partial \Psi}{\partial t} + \mathbf{V} \cdot \nabla \Psi = \epsilon^2 \nabla^2 \Psi - F'(\Psi) + \beta(t) k(\Psi) \tag{125}$$

$$F(\Psi) = 0.25(\Psi^2 - 1)^2 \tag{126}$$

$$F'(\Psi) = \Psi(\Psi^2 - 1) \tag{127}$$

$$k(\Psi) = 2\sqrt{F(\Psi)} \tag{128}$$

$$\beta(t) = \frac{\int_\Omega F'(\Psi(\mathbf{x}, t)) d\mathbf{x}}{\int_\Omega 2\sqrt{F(\Psi(x, t))} d\mathbf{x}} \tag{129}$$

The Allen-Cahn equations for solidification

$$\tau \frac{\partial \Psi}{\partial t} = \frac{\partial}{\partial y}\Big(\epsilon \frac{\partial \epsilon}{\partial \theta} \frac{\partial \Psi}{\partial x}\Big) - \frac{\partial}{\partial x}\Big(\epsilon \frac{\partial \epsilon}{\partial \theta} \frac{\partial \Psi}{\partial y}\Big) + \nabla \cdot (\epsilon^2 \nabla \Psi) + \Psi(1 - \Psi)\Big(\Psi - \frac{1}{2} + m(T)\Big) \tag{130}$$

$$\epsilon = \epsilon_4 \sigma(\theta) \tag{131}$$

$$\sigma(\theta) = 1 + \delta cos(j(\theta - \theta_0)) \tag{132}$$

$$\theta = tan^{-1}\Big(\frac{\partial \Psi / \partial y}{\partial \Psi / \partial x}\Big) \tag{133}$$

$$m(T) = \Big(\frac{\alpha}{\pi}\Big) tan^{-1}\Big(\gamma(T_{eq} - T)\Big) \tag{134}$$

$$\frac{\partial T}{\partial t} = \nabla^2 T + L \frac{\partial \Psi}{\partial t} \tag{135}$$

use $\tau = 0.0003$, $\epsilon_4 = 0.01$, Ł $= 1.8$, $\delta = 0.02$, $j = 4 or 6$, $\alpha = 0.9$, $T_{eq} = 1$, $\theta_0 = 0.2$, $dx = dy = 0.03$

The Navier-Stokes Equations

$$\frac{\partial \rho}{\partial t} = \nabla \cdot (\rho \mathbf{V}) \quad \leftarrow \textit{continuitiy equation} \tag{136}$$

for incompressible flow:

$$0 = \nabla \cdot \mathbf{V} \quad \leftarrow \textit{continuitiy equation} \tag{137}$$

$$\frac{\partial \mathbf{V}}{\partial t} = \nu \nabla^2 \mathbf{V} - \mathbf{V} \cdot \nabla \mathbf{V} - \frac{1}{\rho} \nabla P + \mathbf{F} \quad \leftarrow \textit{momentum equation} \tag{138}$$

$$\rho C_p \frac{\partial T}{\partial t} = K \nabla^2 T - \mathbf{V} \cdot \nabla T + Q(T) \quad \leftarrow \textit{energy equation} \tag{139}$$

$$\frac{\partial C}{\partial t} = D \nabla^2 C - \mathbf{V} \cdot \nabla C + J(C) \quad \leftarrow \textit{species transport equation} \tag{140}$$

where ρ is the material density, t is time, $\mathbf{V}$ is fluid velocity vector, μ is fluid viscosity, P is pressure, F is external forces, T is temperature, K is thermal conductivity, Q is energy generation/absorption, C_p is the thermal heat capacity, C is the concentration of solvent or solute, D is the chemical diffusivity of solvent or solute, J is the chemical reactivity term.

Dimensionless Cahn-Hilliard-Navier-Stokes Equations

$$\nabla \cdot \mathbf{V} = 0 \quad \leftarrow \textit{continuitiy equation} \tag{141}$$

$$\frac{\partial \mathbf{V}}{\partial t} + \mathbf{V} \cdot \nabla \mathbf{V} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho Re}\nabla^2 \mathbf{V} \quad \leftarrow \textit{momentum equation} \tag{142}$$

$$\frac{\partial \Psi}{\partial t} + \nabla \cdot (\Psi \mathbf{V}) = \frac{1}{Pe}\nabla^2 \mu \tag{143}$$

$$\mu = \Psi^3 - \Psi - ^2 \nabla^2 \Psi \tag{144}$$

$$Re = \frac{\rho_c u_c L_c}{\eta} \tag{145}$$

$$We = \frac{\rho_c L_c u_c^2}{\sigma} \tag{146}$$

$$Fr = \frac{u_c^2}{g L_c} \tag{147}$$

$$Pe = \frac{L_c u_c}{M \mu_c} \tag{148}$$

Allen-Cahn-Navier-Stokes Equations

$$\nabla \cdot \mathbf{V} = 0 \quad \leftarrow \textit{continuitiy equation} \tag{149}$$

$$\frac{\partial \mathbf{V}}{\partial t} + \mathbf{V} \cdot \nabla \mathbf{V} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{V} + \mathbf{F} \quad \leftarrow \textit{momentum equation} \tag{150}$$

$$\rho C_p \frac{T}{\partial t} + \rho C_p \mathbf{V} \cdot \nabla T = \nabla \cdot (k \nabla T) + Q(T) \quad \leftarrow \textit{energy equation} \tag{151}$$

$$\frac{\partial \Psi}{\partial t} + \nabla \cdot (\Psi \mathbf{V}) = \nabla \cdot M \left(\nabla \Psi - \frac{1 - 4(\Psi - \Psi_0)^2}{\epsilon} \mathbf{n} \right) \quad \leftarrow \textit{Allen-Cahn equation} \tag{152}$$

$$\mathbf{F}_s = \mu_\Psi \nabla \Psi \quad \leftarrow \textit{surface tension force} \tag{153}$$

$$\mu_\Psi = 1.5\sigma[32\Psi(\Psi - 1)(\Psi - 0.5)/\epsilon - \epsilon \nabla^2 \Psi] \quad \leftarrow \textit{chemical potential} \tag{154}$$

$$\mathbf{F}_b = \rho \mathbf{g} \quad \leftarrow \textit{body force} \tag{155}$$

$$\mathbf{n} = \frac{\nabla \Psi}{|\nabla \Psi|} \quad \leftarrow \textit{normal vector} \tag{156}$$

$$\Psi(x) = \Psi_0 \pm \frac{\Psi_H - \Psi_L}{2} tanh\Big(\frac{|\mathbf{x} - \mathbf{x}_0|}{0.5\epsilon} \Big) \tag{157}$$

if $\Psi_H = 1$ for the heavy fluid, $\Psi_L = 0$ for the light fluid and $\Psi_0 = 0.5$ for the interface then:

$$\Psi(x) = 0.5\Big[1 - tanh\Big(\frac{|\mathbf{x} - \mathbf{x}_0|}{0.5\epsilon} \Big)\Big] \tag{158}$$

$$At = \frac{\rho_H - \rho + L}{\rho_H + \rho L} \quad \leftarrow \textit{Atwood number} \tag{159}$$

$$Re = \frac{\rho_H u_0 L}{\mu_H} \quad \leftarrow \textit{Reynolds number} \tag{160}$$

$$u_0 = \sqrt{gL} \quad \leftarrow \textit{reference velocity scale} \tag{161}$$

$$Ca = \frac{\mu_H u_0}{\sigma} \quad \leftarrow \textit{Cabilary number} \tag{162}$$

$$Pe = \frac{u_0 L}{M} \quad \leftarrow \textit{Peclet number} \tag{163}$$

The Volume of Fluid Method for Boiling

$$\rho = \Phi\rho_l + (1-\Phi)\rho_v \tag{164}$$

$$\frac{\partial \Phi}{\partial t} + \nabla \cdot (\Phi \mathbf{V}) = -m^{\cdot}\frac{1}{\rho_l} \tag{165}$$

where $m^{\cdot}$ is the mass transfer rate per Unit volume obtained from the phase change model and is posotive for boiling and negative for condensation. The interfacial mass transfer rate ($kg/m^2 s$) is expressed as:

$$M^{\cdot} = (q_l + q_v)/L \tag{166}$$

where q_l and q_v are the heat fluxes (W/m^2) coming from the liquid and vapor sides of the interface. and L is the latent heat of vaporization. The heat fluxes are expressed as:

$$q_l = \lambda_l \nabla T_l \cdot \vec{n} \tag{167}$$

$$q_l = -\lambda_v \nabla T_v \cdot \vec{n} \tag{168}$$

The mass transfer rate per Unit volume ($kg/m^3 s$) is expressed as:

$$m^{\cdot} = M^{\cdot} S_{int}/V_{cell} \tag{169}$$

The interface sharpening equation can be expressed as

$$\frac{\partial \Phi}{\partial \tau} + \beta \nabla \cdot (\Phi(1-\Phi)\vec{n}) = \beta \nabla \cdot (\epsilon \nabla \Phi) \tag{170}$$

Hele-Shaw Flow Equations

$$\nabla^2 P = 0 \qquad \forall \mathbf{x} \in \Omega \tag{171}$$

$$P = -\gamma\kappa \qquad \forall \mathbf{x} \in \Gamma \tag{172}$$

$$-\frac{b^2}{12\mu}\frac{\partial P}{\partial n} = V_n \qquad \forall \mathbf{x} \in \Gamma \tag{173}$$

where b is the gap width, μ is the fluid viscosity

Sharp-Interface Phase Transformation in Binary Systems with Convection

$$\frac{\partial V_x}{\partial t} + V_x\frac{\partial V_x}{\partial x} + V_y\frac{\partial V_x}{\partial y} + V_z\frac{\partial V_x}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial x} = \nu\Big(\frac{\partial^2 V_x}{\partial x^2} + \frac{\partial^2 V_x}{\partial y^2} + \frac{\partial^2 V_x}{\partial z^2}\Big) + \rho g_x \tag{174}$$

$$\frac{\partial V_y}{\partial t} + V_x\frac{\partial V_y}{\partial x} + V_y\frac{\partial V_y}{\partial y} + V_z\frac{\partial V_y}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial y} = \nu\Big(\frac{\partial^2 V_y}{\partial x^2} + \frac{\partial^2 V_y}{\partial y^2} + \frac{\partial^2 V_y}{\partial z^2}\Big) + \rho g_y \tag{175}$$

$$\frac{\partial V_z}{\partial t} + V_x\frac{\partial V_z}{\partial x} + V_y\frac{\partial V_z}{\partial y} + V_z\frac{\partial V_z}{\partial z} + \frac{1}{\rho}\frac{\partial p}{\partial z} = \nu\Big(\frac{\partial^2 V_z}{\partial x^2} + \frac{\partial^2 V_z}{\partial y^2} + \frac{\partial^2 V_z}{\partial z^2}\Big) + \rho g_z \tag{176}$$

$$\rho C_p\frac{\partial T}{\partial t} + V_x\frac{\partial T}{\partial x} + V_y\frac{\partial T}{\partial y} + V_z\frac{T}{\partial z} = K\Big(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\Big) + Q(T) \tag{177}$$

$$\frac{\partial C}{\partial t} + V_x\frac{\partial C}{\partial x} + V_y\frac{\partial C}{\partial y} + V_z\frac{C}{\partial z} = D\Big(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} + \frac{\partial^2 C}{\partial z^2}\Big) + R(C) \tag{178}$$

$$\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z} = 0 \tag{179}$$

$$T_i = T_m - \kappa\epsilon_C - V_n\epsilon_V \tag{180}$$

$$C_i = C_0 + \frac{T_i - T_m + \Gamma\kappa\sigma(\theta,\theta_o)}{m_{liq}} \tag{181}$$

$$\kappa = \nabla.\frac{\nabla\Phi}{|\nabla\Phi|} \tag{182}$$

$$\epsilon_C(\mathbf{x}) = d_o(1 - Acos(k_A\theta + \theta_o)) \tag{183}$$

$$\epsilon_V(\mathbf{x}) = d_o(1 - Acos(k_A\theta + \theta_o)) \tag{184}$$

$$V_n = \frac{1}{L}\left(K_S\frac{\partial T_S}{\partial \mathbf{n}} - K_L\frac{\partial T_L}{\partial \mathbf{n}}\right) \tag{185}$$

$$V_n = \frac{1}{C_{liq} - C_{sol}}\left(D_S\frac{\partial C_S}{\partial \mathbf{n}} - D_L\frac{\partial C_L}{\partial \mathbf{n}}\right) \tag{186}$$

$$\mathbf{n} = -\frac{\nabla\Phi}{|\nabla\Phi|} \tag{187}$$

$$\frac{\partial\Phi}{\partial t} + V_n^{ext}|\nabla\Phi| = 0 \tag{188}$$

$$\frac{\partial f_{sol}}{\partial t} + V_n^{ext}|\nabla f_{sol}| = 0 \tag{189}$$

$$\nabla^2 V_n^{ext} = 0, \quad V_n^{ext}|_\Gamma = V_n \tag{190}$$

$$C_{solidus} = kC_{liquidus} \tag{191}$$

$$Q(T) = -\rho L \frac{\partial f_{sol}}{\partial t} \tag{192}$$

$$R(C) = C(1-k)\frac{\partial f_{sol}}{\partial t} \tag{193}$$

The Projection Method

$$\frac{\mathbf{V}^* - \mathbf{V}^t}{\Delta t} = \nu\nabla^2\mathbf{V} - \mathbf{V}\cdot\nabla\mathbf{V} + \mathbf{F} \tag{194}$$

$$\nabla^2 P = -\frac{\rho}{\Delta t}\Big(\nabla\cdot\mathbf{V}^*\Big) \tag{195}$$

$$\frac{\mathbf{V}^{t+1} - \mathbf{V}^*}{\Delta t} = -\frac{1}{\rho}\nabla P \tag{196}$$

The Artificial Compressibility Method

$$\frac{\partial \rho}{\partial t} + \rho\Big(\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}\Big) = 0 \tag{197}$$

having relaxed the incompressibility constraint:

$$\frac{\partial \rho}{\partial P}\frac{\partial P}{\partial t} + \rho\Big(\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}\Big) = 0 \tag{198}$$

upon rearranging:

$$\frac{\partial P}{\partial t} + \rho C^2\Big(\frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}\Big) = 0 \tag{199}$$

where

$$\frac{1}{C^2} = \frac{\partial \rho}{\partial P} \tag{200}$$

where

$$C = \beta\sqrt{max\Big((V_x^2 + V_y^2 + V_z^2), 0.5V_{ref}^2\Big)} \tag{201}$$

Differential Operators

$$grad = \nabla = \begin{bmatrix} \partial/\partial x \\ \partial/\partial y \\ \partial/\partial z \end{bmatrix} \tag{202}$$

$$grad \quad f = \nabla f = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \\ \partial f/\partial z \end{bmatrix} \tag{203}$$

$$div\mathbf{V} = \nabla \cdot \mathbf{V} = \nabla \cdot \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{204}$$

$$curl\mathbf{V} = \nabla \times \mathbf{V} = \nabla \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \partial w/\partial y - \partial v/\partial z \\ \partial u/\partial z - \partial w/\partial x \\ \partial v/\partial x - \partial u/\partial y \end{bmatrix} \tag{205}$$

$$\Delta = \nabla^2 = \nabla \cdot \nabla = \frac{\partial}{\partial x^2} + \frac{\partial}{\partial y^2} + \frac{\partial}{\partial z^2} \tag{206}$$

Differential Identities

$$f\frac{\partial \rho}{\partial t} = \frac{\partial (f\rho)}{\partial t} - \rho\frac{\partial f}{\partial t} \tag{207}$$

$$f\nabla \cdot (\rho \mathbf{V}) = \nabla \cdot (f\rho\mathbf{V}) - \rho\mathbf{V} \cdot \nabla f \tag{208}$$

Normal vector:

$$\mathbf{n} = \frac{\nabla\Phi}{|\nabla\Phi|} \tag{209}$$

Normal velocity:

$$\mathbf{n} \cdot \mathbf{V} = V_n = -\frac{1}{|\nabla\Phi|}\frac{\partial\Phi}{\partial t} \tag{210}$$

Curvature:

$$\kappa = \nabla \cdot \mathbf{n} = \nabla \cdot \frac{\nabla\Phi}{|\nabla\Phi|} = -\frac{1}{|\nabla\Phi|}\Big(\nabla^2\Phi - \frac{(\nabla\Phi\nabla)|\nabla\Phi|}{|\nabla\Phi|}\Big) \tag{211}$$

$$\kappa = \frac{1}{|\nabla\Phi|}\Big(\nabla^2\Phi + \frac{\Phi(1-\Phi^2)}{\epsilon^2}\Big) \tag{212}$$

---

6 Running PntWrks3D

To run FPM3D first you'll need a Unix-like operating system. The code has been tested on Debian and FreeBSD but technically any Unix-like operating system that has bash and gfortran should be able to compile the code.

6.1 Running on Debian

6.2 Running on FreeBSD