2005-01

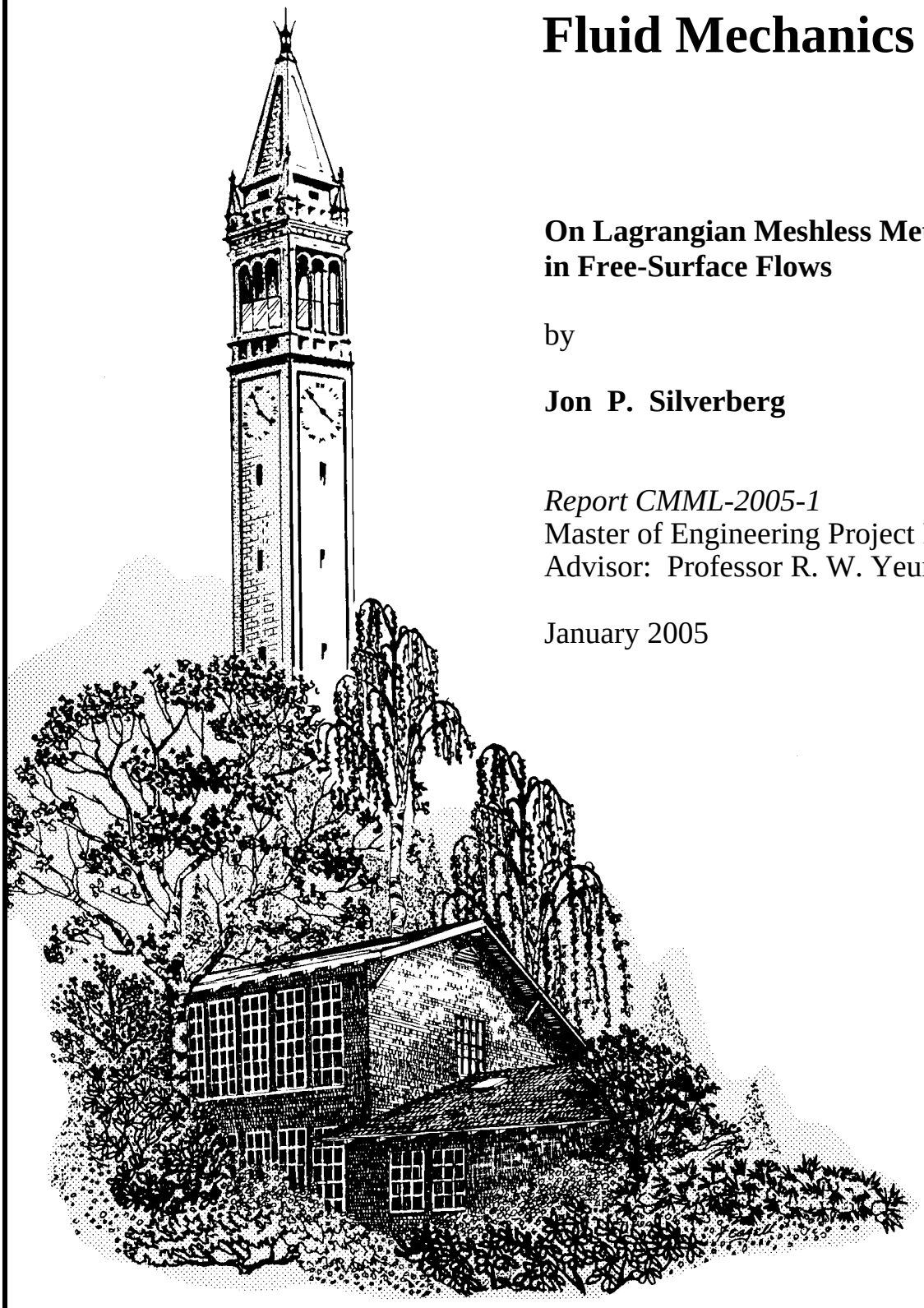
On Lagrangian meshless methods in free-surface flows

Silverberg, Jon P.

Monterey, California. Naval Postgraduate School

COLLEGE OF ENGINEERING

MECHANICAL ENGINEERING
Fluid Mechanics



On Lagrangian Meshless Methods in Free-Surface Flows

by

Jon P. Silverberg

Report CMML-2005-1
Master of Engineering Project Report
Advisor: Professor R. W. Yeung

January 2005

UNIVERSITY OF CALIFORNIA AT BERKELEY, CA 94720-1740
Computational Marine Mechanics Laboratory Tel. (510) 642-8347 Telefax (510) 642-5539

On Lagrangian Meshless Methods in Free-Surface Flows

By

Jon P. Silverberg

INDIVIDUAL RESEARCH PROJECT

Submitted in partial satisfaction of the requirements for the

degree of

MASTER OF ENGINEERING

in

Ocean Engineering

in the

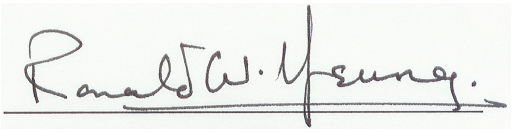
GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA AT BERKELEY

December 2004

Approved by thesis supervisor:



Professor Ronald W. Yeung
Department of Mechanical Engineering
and Ocean Engineering Graduate Group

Abstract

Classically, fluid dynamics have been dealt with analytically because of the lack of numerical resources (Yeung, 1982). With the development of computational ability, many formulations have been developed which typically use the traditional Navier-Stokes equations along with an Eulerian grid. Today, there exists the possibility of using a moving grid (Lagrangian) along with a meshless discretization.

The first issue in meshless fluid dynamics is the equations of motion. There are currently two types of Lagrangian formulations. Spherical Particle Hydrodynamics (SPH) is a method which calculates all equations of motion explicitly. The Moving Particle Semi-implicit (MPS) method uses a mathematical foundation based on SPH. However, instead of calculating all laws of motion explicitly, a fractional time step is performed to calculate pressure. A proposed method, Lagrange Implicit Fraction Step (LIFS), has been created which improves the mathematical formulations on the fluid domain. The LIFS method returns to Continuum mechanics to construct the laws of motion based on decomposing all forces of a volume. It is assumed that all forces on this volume can be linearly superposed to calculate the accelerations of each mass. The LIFS method calculates pressure from a boundary value problem with the inclusion of proper flux boundary conditions.

The second issue in meshless Lagrangian dynamics is the calculation of derivatives across a domain. The Monte Carlo Integration (MCI) method uses weighted averages to calculate operators. However, the MCI method can be very inaccurate, and is not suitable for sparse grids. The Radial Basis Function (RBF) method is introduced and studied as a possibility to calculate meshless operators. The RBF method involves a solution of a system of equations to calculate interpolants. Machine expenses are shown to limit the viability of the RBF method for large domains. A new method of calculation has been created called Multi-dimensional Lagrange Interpolating Polynomials (MLIP). While Lagrange Interpolating Polynomials are essentially a one-dimensional interpolation, the use of "dimensional-cuts" and Gaussian quadratures can provide multi-dimensional interpolation.

This paper is divided into three sections. The first section specifies the equations of motion. The second section provides the mathematical basis for meshless calculations. The third section evaluates the effectiveness of the meshless calculations and compares two fluid-dynamic codes.

Acknowledgments

I would like to express gratitude to my advisor, Professor Ronald Yeung. His knowledge of fluid dynamics and his support of my ideas were invaluable in allowing me to complete this research. I also thank the Society of Naval Architects and Marine Engineers for support in my studies at Berkeley. At Berkeley, Professors Zohdi, Mansour, and Morris from the Mechanical Engineering Department were very helpful for providing me with the mathematical background required for this subject.

I am also grateful for the continued support of my Professors from the United States Naval Academy. I thank Professor Paul Miller for teaching me how to complete a large research project. I also thank Professor Bruce Johnson for giving me initial interest into meshless fluid dynamics and for his subsequent advice throughout my studies.

Finally, I would like to thank my colleague Lars Flæten for his friendship and ideas as we both pursued our individual research. Finally, I would also like to thank my parents and my girlfriend, Tiffany, for their support.

Contents

List of Figures

List of Tables

Chapter 1

Equations of Motion

Eulerian fluid dynamics calculates the flow of fluid through a given volume. Lagrangian dynamics calculates the movement of discrete volumes in a flow. We will utilize Lagrangian forms of equations to model highly deformable flows, while tracking the movement of discrete volumes. All types of Lagrangian formulations will use a variation of the Navier-Stokes equations.

1.1 Navier-Stokes Equations

Continuum mechanics provides a context to construct the laws of motion. We will use balance laws of linear momentum, angular momentum, and mass (Chadwick, 1999). The basic postulate for linear momentum on a material volume is from Newton's second law (Johnson, 1990):

> There exists a frame of reference for which, at any instant, the rate of change of linear momentum of a material volume, Ω, is equal to the resultant force acting on the mass in that volume.

$$\sum \underline{F} = \underline{F}_{\partial\Omega} + \underline{F}_{\Omega} = \frac{\mathcal{D}}{\mathcal{D}t}\iiint_{\Omega} \rho \underline{u} \mathrm{d}\Omega \tag{1.1}$$

in which $\underline{F}$ is a force vector, $\partial\Omega$ is the surface of the material volume, $\frac{\mathcal{D}}{\mathcal{D}t}$ is the material derivative, ρ is the density of the fluid, and $\underline{u}$ is the velocity vector.

Cauchy developed the idea of a stress tensor, $\underline{\underline{\sigma}}$, in which:

$$\underline{F}_{\partial\Omega} = \iint_{\partial\Omega} \underline{\underline{\sigma}} \cdot \underline{n} \mathrm{d}S \qquad \underline{F}_{\Omega} = \iiint_{\Omega} \underline{f} \mathrm{d}\Omega \tag{1.2}$$

where $\underline{n}$ is the outward unit normal and $\underline{f}$ are the body forces. By use of the divergence theorem we can relate the surface integral to a volume integral.

$$\iint_{\partial\Omega} \underline{\underline{\sigma}} \cdot \underline{n} \mathrm{d}S = \iiint_{\Omega} \nabla \cdot \underline{\underline{\sigma}} \mathrm{d}\Omega \tag{1.3}$$

Therefore, our balance of linear momentum can be expressed as:

$$\frac{\mathcal{D}}{\mathcal{D}t}\iiint_{\Omega}\rho\underline{u}\mathrm{d}\Omega=\iiint_{\Omega}\nabla\cdot\underline{\underline{\sigma}}\mathrm{d}\Omega+\iiint_{\Omega}\underline{f}\mathrm{d}\Omega \tag{1.4}$$

Dealing with only volume integrals, the integrand must vanish within the fluid, and we get Cauchy's equation of motion:

$$\frac{\mathcal{D}}{\mathcal{D}t}\rho\underline{u}=\nabla\cdot\underline{\underline{\sigma}}+\underline{f} \tag{1.5}$$

To locally conserve angular momentum, we specify that the stress tensor is symmetric.

$$\underline{\underline{\sigma}}=\underline{\underline{\sigma}}^{\mathsf{T}} \tag{1.6}$$

Finally, we will specify conservation of mass.

$$\frac{\mathcal{D}}{\mathcal{D}t}\iiint_{\Omega}\rho\mathrm{d}\Omega=0 \tag{1.7}$$

Using the transport theorem and specifying that the integrand is zero throughout the fluid, we can express conservation of mass as:

$$\frac{\mathcal{D}\rho}{\mathcal{D}t}+\rho\nabla\cdot\underline{u}=0 \tag{1.8}$$

1.1.1 Constitutive relations

We will specify a Newtonian constitutive relation (Newman, 1977).

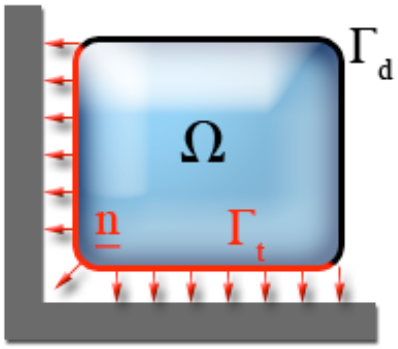
$$\underline{\underline{\sigma}}=-p\underline{\underline{I}}+\mu\underline{\underline{D}} \tag{1.9}$$

in which p is the pressure, $\underline{\underline{I}}$ is the identity tensor, μ is the viscosity coefficient, and $\underline{\underline{D}}$ is the symmetric deformation tensor.

$$[\sigma]=\begin{bmatrix}-p&0&0\\0&-p&0\\0&0&-p\end{bmatrix}+\mu\begin{bmatrix}2\frac{\partial u}{\partial x}&\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}&\frac{\partial u}{\partial z}+\frac{\partial w}{\partial x}\\\frac{\partial v}{\partial x}+\frac{\partial u}{\partial y}&2\frac{\partial v}{\partial y}&\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\\\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}&\frac{\partial w}{\partial y}+\frac{\partial v}{\partial z}&2\frac{\partial w}{\partial z}\end{bmatrix} \tag{1.10}$$

1.1.2 Boundary Conditions

Our fluid domain (Ω) is shown in figure (1.1). There are two boundary conditions on the domain. Dirichlet boundary conditions apply to the free surface. Neumann boundary conditions apply to the surface next to the wall. The walls are not a part of the fluid domain, but serve to administer the Neumann boundary conditions.



A diagram of the domain

- The fluid domain Ω is shown in blue
- There are Neumann boundaries Γ_t and Dirichlet boundaries Γ_d
- There is closure such that $\overline{\partial\Omega} = \overline{\Gamma_t \cup \Gamma_d}$
- Outward unit normals are shown as $\underline{n}$

Figure 1.1: Domain used in equations of motion

1.1.3 Lagrangian Forms

Part of the ease of Lagrangian coordinates is being able to keep the material derivatives instead of using a convection term. In doing so, we express the equations of motion for discrete volumes of fluid. These individual volumes move with the flow of the fluid. If we enforce water fluid properties such that $\frac{\mathcal{D}\rho}{\mathcal{D}t} = 0$, then the law of motion is simply: (Brodkey, 1995)

$$\underline{a} = \frac{1}{\rho}\left(\nabla \cdot \underline{\underline{\sigma}} + \underline{f}\right) \tag{1.11}$$

where $\underline{a}$ is the acceleration vector which is equal to the material derivative of velocity. If we use explicit time-stepping, we can obtain:

$$\delta\underline{u} = \frac{\delta t}{\rho}\left(\nabla \cdot \underline{\underline{\sigma}} + \underline{f}\right) \tag{1.12}$$

in which,

$$\underline{u}^{n+1} = \underline{u}^{n} + \delta\underline{u} \tag{1.13}$$

Position vectors are therefore:

$$\underline{x}^{n+1} = \underline{x}^{n} + \delta t\underline{u}^{n+1} \tag{1.14}$$

The continuity law is:

$$-\rho\left(\nabla \cdot \underline{u}\right) = 0 \tag{1.15}$$

1.2 Spherical Particle Hydrodynamics

Monaghan first suggested the use of individual particles in the calculation of fluid dynamics. Each particle interacts with neighboring particles according to the law of motion. (Monaghan, 1994)

SPH is calculated in a fully explicit manner. Using eqn(1.11), we calculate $\underline{\underline{\sigma}}$ explicitly as a function of particles a and b:

$$-\underline{\underline{\sigma}}_{ab} = \frac{p_a}{\rho_a^2} + \frac{p_b}{\rho_b^2} + \Pi_{ab} \tag{1.16}$$

Pressure is calculated as an equation of state for a very "stiff" gas. It is a function of the density calculated at a time step n versus the standard density (ρ^0):

$$p_a^n = B\left(\left(\frac{\rho_a^n}{\rho_a^0}\right)^\gamma - 1\right) \tag{1.17}$$

The parameter γ is set arbitrarily at 7, and the coefficient B is somewhat defined by the speed of sound. Density is calculated for each particle as:

$$\rho_a = \int m_a \mathrm{d}\Omega_a \tag{1.18}$$

Both bulk and shear viscosity are calculated through:

$$\Pi_{ab} = \frac{-\alpha c \mu_{ab} + \beta \mu_{ab}^2}{1/2\left(\rho_a + \rho_b\right)} \tag{1.19}$$

The parameter μ_{ab} roughly tries to simulate the laplacian operator, α introduces the viscosity, c is the speed of sound, and β is usually set to zero.

Boundary conditions for SPH are treated in an ad-hoc approach. Boundary particles are treated no differently than any other particle, regardless of if it is on the free surface or is next to a wall. The most common way to ensure that particles in SPH do not violate wall boundaries is through an extra force. When a particle comes near a wall, a repulsive force pushes on the particle such that it does come into the wall. The degree of force required, however, must be set for different types of simulations.

The basic algorithm for this method is shown in table 1.1.

There have been very good results with the SPH equations including dam collapse and breaking waves (Monaghan, 1994). Furthermore, SPH has been shown to be robust in "floating object" problems including a falling wedge penetrating water as well as a wave interaction with a floating box (Doring et al., 2002).

INITIALIZE the domain	
CALCULATE the initial density, (ρ^0)	eqn(1.18)
FOR EACH time step, n	
CALCULATE the density, (ρ^n)	eqn(1.18)
EVALUATE the pressure, (p^n)	eqn(1.17)
EVALUATE the viscosity, (Π^n)	eqn(1.19)
EVALUATE the motion, $\delta\underline{u}$	eqn(1.12)
UPDATE the velocity, $(\underline{u}^{n+1})$	eqn(1.13)
UPDATE the position, $(\underline{x}^{n+1})$	eqn(1.14)
END	

Table 1.1: SPH Algorithm

1.3 Moving Particle Semi-Implicit

The MPS equations are a variation on SPH in that instead of pressure being calculated explicitly, it is solved via Poisson's equation (Yoon et al., 1999). This is done by a fractional step which splits the laws of motion into two parts and uses the continuity equation to connect the two.

The first fractional step is defined as:

$$\underline{u}^* = \underline{u}^n + \frac{\delta t}{\rho}\left(\nu\nabla^2\underline{u}^n + \rho\underline{g}\right) \tag{1.20}$$

The second fractional step is defined as:

$$\underline{u}^{**} = -\frac{\delta t}{\rho}\nabla p^{n+1} \tag{1.21}$$

In order to calculate the second fractional step, we must find a way to calculate the pressure. This can be done by first finding the change in density due to the first fractional step

$$\frac{\partial\rho^*}{\partial t} = -\frac{1}{\delta t}\frac{\rho^* - \rho^0}{\rho^0} \tag{1.22}$$

Pressure is calculated as a solution to Poisson's equation. There is only one boundary condition in which particles on the free surface have Dirichlet conditions where the pressure is set to zero. Particles near walls do not have a boundary condition, but are instead treated by the field equation. The right hand side of the field equation is the divergence of the first fractional velocity:

$$\nabla^2 p^{n+1} = \frac{\rho}{\delta t}\frac{\partial\rho^*}{\partial t} \tag{1.23}$$

Finally, the total velocity is the summation of fractional velocities:

$$\underline{u}^{n+1} = \underline{u}^{*} + \underline{u}^{**} \tag{1.24}$$

Boundary conditions in MPS have more treatment than in SPH. First, particles on the free surface have the pressure set to zero. Particles next to the walls, however, do not have any boundary conditions. Instead, the walls are treated as fluid particles which are stationary. This procedure is easy to implement, however it violates the equations of motion. The laws of motion are expressed in terms of integrals over a fluid domain as well as material derivatives. When a derivative is calculated which is not material, than the equation of motion is invalid.

The basic algorithm for this method is shown in table 1.2.

INITIALIZE the domain	
CALCULATE the initial density, (ρ^0)	eqn(1.18)
FOR EACH time step, n	
EVALUATE the first fractional velocity, $(\underline{u}^*)$	eqn(1.20)
CALCULATE the density, (ρ^*)	eqn(1.18)
CALCULATE the change in density, $(\frac{\partial \rho^*}{\partial t})$	eqn(1.22)
SOLVE for pressure, (p^{n+1})	eqn(1.23)
EVALUATE the second fractional velocity, $(\underline{u}^{**})$	eqn(1.21)
SUMMATE the fractional velocities, $(\underline{u}^{n+1})$	eqn(1.24)
UPDATE the position, $(\underline{x}^{n+1})$	eqn(1.14)
END	

Table 1.2: MPS Algorithm

Again, good results have been obtained for a sloshing tank (Yoon et al., 1999), breaking waves (Lo and Shao, 2002), and wave generation (Gotoh and Sakai, 1999). However, it should be noted that the differential equation (1.23) only has boundary conditions on one side of the fluid and is therefore an ill-posed boundary value problem.

1.4 Lagrange Implicit Fraction Step

The LIFS equations were created to use the principles of SPH and MPS, but to improve the mathematical foundations. For each force term in the balance of linear momentum, we break it into a corresponding explicit and implicit velocities. We then solve the continuity equation for the implicit velocities which ensures that the momentum equation does not result in a change in density.

To reiterate our explicit procedure, we take the Lagrangian conservation of momentum

and break it into discrete time steps:

$$\underline{a} = \frac{1}{\rho}\left(\nabla \cdot \underline{\underline{\sigma}} + \underline{f}\right) \tag{1.25}$$

$$\delta\underline{u} = \frac{\delta t}{\rho}\left(\nabla \cdot \underline{\underline{\sigma}} + \underline{f}\right) \tag{1.26}$$

$$\underline{u}^{n+1} = \underline{u}^n + \delta\underline{u} \tag{1.27}$$

We assume that the constitutive relation eqn(1.9) and our summation of forces is linear:

$$\underline{\underline{\sigma}} = \underbrace{-p\underline{\underline{I}}}_{\mathring{\underline{\underline{\sigma}}}} + \underbrace{\mu\underline{\underline{D}}}_{\hat{\underline{\underline{\sigma}}}} \tag{1.28}$$

For each time step, we will define intermediate velocities based on the forces which contributed to them:

$$\delta\underline{u} = \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}} + \delta\mathring{\underline{u}} \tag{1.29}$$

The tilde velocity is the motion due to body forces, the hat velocity is the motion due to viscosity, and the spherical velocity is the motion due to pressure.

$$\delta\widetilde{\underline{u}} = \frac{\delta t}{\rho}\underline{f} \tag{1.30}$$

$$\delta\hat{\underline{u}} = \frac{\delta t}{\rho}\nabla \cdot \hat{\underline{\underline{\sigma}}} \tag{1.31}$$

$$\delta\mathring{\underline{u}} = \frac{\delta t}{\rho}\nabla \cdot \mathring{\underline{\underline{\sigma}}} \tag{1.32}$$

In each time step, we have both implicit and explicit parts. We define the body forces explicitly and due only to gravity:

$$\delta\widetilde{\underline{u}} = (\delta t)\left(\underline{g}\right) \tag{1.33}$$

The hat velocity is explicitly defined from viscosity calculated from the current velocities:

$$\delta\hat{\underline{u}} = (\delta t)\,\nu\nabla^2\underline{u}^n \tag{1.34}$$

The spherical velocity is calculated implicitly from the pressure (Yeung and Ananthakrishnan, 1992):

$$\delta\mathring{\underline{u}} = -\frac{\delta t}{\rho}\nabla p^{n+1} \tag{1.35}$$

We can obtain the pressure through the continuity equation,

$$\frac{\mathcal{D}\rho}{\mathcal{D}t} + \rho\nabla \cdot \underline{u}^{n+1} = 0 \tag{1.36}$$

$$\frac{\mathcal{D}\rho}{\mathcal{D}t} + \rho\nabla \cdot (\underline{u}^n + \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}} + \delta\mathring{\underline{u}}) = 0 \tag{1.37}$$

Assuming ρ is constant in time, we separate the explicit and implicit velocity terms (Ananthakrishnan and Yeung, 1994):

$$\nabla \cdot \delta\mathring{\underline{u}} = -\nabla \cdot (\underline{u}^n + \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}}) \tag{1.38}$$

By applying our constitutive relation into our spherical velocity, the continuity equation becomes:

$$\nabla \cdot \left(\frac{\delta t}{\rho} \left(\nabla p^{n+1} \right) \right) = \nabla \cdot (\underline{u}^n + \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}}) \tag{1.39}$$

Because we are working with incompressible fluid, we can show that:

$$\nabla \cdot \underline{u}^n = 0 \tag{1.40}$$

Furthermore, since the body forces are simply a constant scalar field:

$$\nabla \cdot \delta\widetilde{\underline{u}} = 0 \tag{1.41}$$

Therefore, the only term left is due to viscosity. This can be found to be Poisson's equation:

$$\nabla^2 p^{n+1} = \frac{\rho}{\delta t} \nabla \cdot \delta\hat{\underline{u}} = \frac{1}{\delta t} \frac{\partial \hat{\rho}}{\partial t} \tag{1.42}$$

This can be setup as a boundary value problem by making specifications on the field equations and the boundary conditions:

$$\boxed{\begin{array}{l} 1)\ \nabla^2 p^{n+1} = \dfrac{\rho}{\delta t} \nabla \cdot \delta\hat{\underline{u}} \text{ in } \Omega \\ 2)\ p^{n+1} = 0 \text{ on } \Gamma_d, \text{ the free surface} \\ 3)\ \underline{u}^{n+1} \cdot \underline{\mathrm{n}} = 0 \text{ on } \Gamma_t, \text{ the walls} \end{array}} \tag{1.43}$$

The third prescription can be found through:

$$\begin{aligned} \underline{u}^{n+1} \cdot \underline{\mathrm{n}} &= 0 \\ (\underline{u}^n + \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}} + \delta\mathring{\underline{u}}) \cdot \underline{\mathrm{n}} &= 0 \\ \delta\mathring{\underline{u}} \cdot \underline{\mathrm{n}} &= -(\underline{u}^n + \delta\widetilde{\underline{u}} + \delta\hat{\underline{u}}) \cdot \underline{\mathrm{n}} \end{aligned} \tag{1.44}$$

Now, we insert eqn(1.35) into eqn(1.44). We express the directional derivative in the $\underline{\mathrm{n}}$ direction as: $\nabla_{\underline{\mathrm{n}}}$.

$$\nabla_{\underline{\mathrm{n}}} p^{n+1} = \frac{\rho}{\delta t} (\underline{u}^n \cdot \underline{\mathrm{n}} + \delta\widetilde{\underline{u}} \cdot \underline{\mathrm{n}} + \delta\hat{\underline{u}} \cdot \underline{\mathrm{n}}) \tag{1.45}$$

We can also simplify this equation in that $\underline{u}^n \cdot \underline{\mathrm{n}} = 0$. Therefore, our boundary condition is:

$$\nabla_{\underline{\mathrm{n}}} p^{n+1} = \frac{\rho}{\delta t} (\delta\widetilde{\underline{u}} \cdot \underline{\mathrm{n}} + \delta\hat{\underline{u}} \cdot \underline{\mathrm{n}}) \tag{1.46}$$

An algorithm for the Lagrange Implicit Fraction Step method is shown in table 1.3.

INITIALIZE the domain	
FOR EACH time step, n	
EVALUATE the body forces for $\delta\widetilde{\underline{u}}$	eqn(1.33)
EVALUATE the viscous forces for $\delta\hat{\underline{u}}$	eqn(1.34)
EVALUATE the divergence for the BVP	eqn(1.42)
EVALUATE the flux BCs for the BVP	eqn(1.46)
SOLVE Poisson's equation for p^{n+1}	eqn(1.43)
EVALUATE the spherical forces for $\delta\mathring{\underline{u}}$	eqn(1.35)
EVALUATE the motion, $(\delta\underline{u})$	eqn(1.29)
UPDATE the velocity, $(\underline{u}^{n+1})$	eqn(1.13)
UPDATE the position, $(\underline{x}^{n+1})$	eqn(1.14)
END	

Table 1.3: LIFS Algorithm

An interesting aspect is when dealing with an inviscid fluid. When this occurs, $\delta\hat{\underline{u}} = 0$. Therefore, the field equation becomes Laplace's equation:

$$\nabla^2 p^{n+1} = 0 \tag{1.47}$$

Furthermore, the flux boundary conditions are only a function of body forces:

$$\nabla_{\underline{\mathrm{n}}} p^{n+1} = \frac{\rho}{\delta t}\delta\widetilde{\underline{u}} \cdot \underline{\mathrm{n}} \tag{1.48}$$

If we set viscosity to zero, we can simplify the algorithm, shown in table 1.4.

INITIALIZE the domain	
FOR EACH time step, n	
EVALUATE the body forces for $\delta\widetilde{\underline{u}}$	eqn(1.33)
EVALUATE the flux BCs for the BVP	eqn(1.48)
SOLVE Laplace's equation for p^{n+1}	eqn(1.47)
EVALUATE the spherical forces for $\delta\mathring{\underline{u}}$	eqn(1.35)
EVALUATE the motion, $(\delta\underline{u})$	eqn(1.29)
UPDATE the velocity, $(\underline{u}^{n+1})$	eqn(1.13)
UPDATE the position, $(\underline{x}^{n+1})$	eqn(1.14)
END	

Table 1.4: LIFS Inviscid Algorithm

Chapter 2

Methods of Computation

After the equations of motion are set, the problem becomes how to calculate the equations between particles. Because all the methods are "meshless," problems of constructing grids are alleviated. However, the question of how particles interact still remain.

The laws of motion, for a meshless method, are extremely dependent on how the particles interact. In essence, the methods of computation represent an interpolation of the solution between particles. The better the interpolation, the better the solution matches the appropriate laws.

For every method of computation, what we wish to find is a continuous function $\hat{f}(\underline{x})$ as a function of discrete values, $f(\underline{x} = \underline{x}_\imath) \equiv f(\underline{x}_\imath)$. Methods of computation typically have the form of finding a continuous function by multiplying a "kernel" (w) by a value (a) evaluated at each discrete location.

$$\hat{f}(\underline{x}) = \sum_{\jmath=0}^{n-1} w_\imath(r_\jmath) a(\underline{x}_\jmath) \tag{2.1}$$

These methods use a nodal-centered position vector as an important value in the weighting function. We will define the radius, $r_{\imath\jmath}$, as the Euclidean norm of two position vectors:

$$r_{\imath\jmath} \equiv \|\underline{x}_\imath - \underline{\xi}_\jmath\| \tag{2.2}$$

Most particle methods evaluate a function g with the origin centered at the particle location $\underline{x}_\imath$. The argument for the function is the distance from another point $\underline{\xi}_\jmath$ to the particle center. We will define the notation for this function g as a function of the radius:

$$g(\underline{x}_\imath, \underline{\xi}_\jmath) \equiv g(r_{\imath\jmath}) \equiv g_\imath(r_\jmath) \tag{2.3}$$

All of these methods of computation are discretely calculated via the use of matrices. For each method of computation, the desired properties of each method includes at least the formation of a matrix which can perform interpolation, (w), as well as matrices which can calculate gradients (dwx and dwy). Furthermore, both the MPS and the LIFS equations

require the calculation of the Laplacian operator. This operator can be represented in matrix form as ddw.

Every one of these matrices required by the equations of motion will be constructed through the use of the following methods. The Monte Carlo Integration (MCI) method calculates both the interpolation and derivatives based on a process of averaging neighbors. The Radial Basis Function (RBF) method calculates these operators by first solving a linear system of equations. After one matrix is inverted, all differencing operations can be performed. Multi-dimensional Lagrange Interpolating Polynomials (MLIP) calculates operators in a meshless manner by creating Lagrange Interpolating Polynomials and their derivatives.

2.1 Monte Carlo Integration

The Monte Carlo Integration method is the traditional method for calculating interpolants in a meshless manner. The principal behind the MCI method is that information about one particle is stored in its neighbors. Therefore, we can use a weighted average of the neighbor particles to not only obtain the interplant, but also to find derivatives.

2.1.1 Formulation

By using an interpolating weighting function, discrete values can be found from the neighbors in the fluid volume (Ω). The weighting kernel w at each node $\imath$ is a function of the distance, $r_\jmath$, and the extent of the kernel, h. The extent of the kernel, h, determines how many other particles are used in the calculations. We will define h as a resolution parameter. As the resolution parameter increases, more neighbor particles are used in the calculations, and the stability of the calculations increases. As the resolution parameter decreases, less neighbor particles are used, and the accuracy of the calculations increases.

$$\hat{f}(\underline{x}) = \int_\Omega f(\underline{\xi}) w_\imath(r_\jmath, h) \mathrm{d}\Omega \tag{2.4}$$

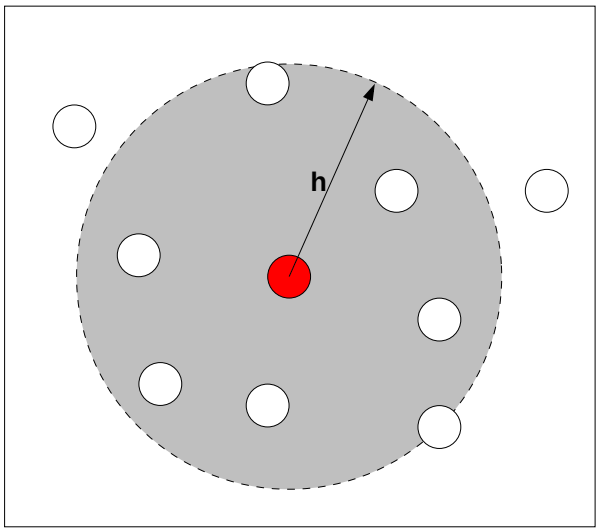
2.1.2 Prescriptions on the kernel

There are three prescriptions on the weighting kernel. The first is that the function, w, is normalized, such that energy is not added nor subtracted from the system.

$$\int_\Omega w_\imath(r_\jmath, h) \mathrm{d}\Omega = 1 \tag{2.5}$$

The second prescription is that the weighting function must become the Dirac delta function as the interaction radius, h, becomes small:

$$\lim_{h \to 0} \int_\Omega f(\underline{\xi}) w_\imath(r_\jmath, h) \mathrm{d}\Omega = f(\underline{x}) \tag{2.6}$$



A diagram of Monte Carlo Integration

- The red particle is the particle to which integration is being performed
- The white particles are neighbor particles
- The length h represents the maximum interaction of the center particle on the surrounding particles

Figure 2.1: Monte Carlo Integration

The third prescription is that the weighting function has a finite radius:

$$\lim_{|r_\jmath|\to h} w_\imath(|r_\jmath|, h) = 0 \tag{2.7}$$

This third prescription allows us to redefine our limits of integration. Instead of the entire fluid domain (Ω), we can redefine the integration to take place only in the domain of each "particle", (Ω_p). Furthermore, if our kernel is smooth and continuous we can take the derivative of $\hat{f}$. Because of finiteness, we find that the operation is passed only to the kernel since $w(h, h) = 0$.

$$\nabla \hat{f}(\underline{x}) = \int_{\Omega_p} f(\underline{\xi}) \nabla w_\imath(r_\jmath, h) \mathrm{d}\Omega_p \tag{2.8}$$

2.1.3 Evaluation

Weighting kernels typically have compact support and are smooth to various degrees of differentiability. Therefore, kernels are mostly based on either splines or Gaussian curves. We will choose a variation on a Gaussian kernel in which:

$$w_\imath(r, h) = \frac{h}{\sqrt{2\pi}} \exp\left(\frac{-r^2}{2h^2}\right) + \frac{r}{2}\left(1 + \mathrm{erf}\left(\frac{r}{\sqrt{2}h}\right)\right) \tag{2.9}$$

$$\nabla_k w(r, h) = \frac{r_k}{2}\left(1 + \mathrm{erf}\left(\frac{r}{\sqrt{2}h}\right)\right) \tag{2.10}$$

$$\nabla^2 w(r, h) = \frac{1}{h\sqrt{2\pi}} \exp\left(\frac{-r^2}{2h^2}\right) \tag{2.11}$$

We can implement the MCI method using three steps. The first step is to initialize the kernel matrices. The second step is to use the kernel to calculate derivatives. The final step is to set up a matrix to invert and solve a linear system.

Table 2.1 shows the initialization routine for the MCI method. All of the matrices for calculating operators are created here: interpolation (W), x-gradient (dWx), y-gradient (dWy), and laplacian (ddW).

```
FOR i = 0 to (n-1)
  FOR j = 0 to (n-1)
    CALCULATE r[i,j]     eqn(2.2)
    CALCULATE W[i,j]     eqn(2.9)
    CALCULATE dWx[i,j]   eqn(2.10)
    CALCULATE dWy[i,j]   eqn(2.10)
    CALCULATE ddW[i,j]   eqn(2.11)
  NEXT j
NEXT i
```

Table 2.1: MCI Initialization Algorithm

Table 2.2 shows how derivatives of a nodally-specified function, f, may be calculated with the operators (dWx) and (dWy). If the kernel was not normalized, we can fix this by dividing the resulting vectors by the sum of the absolute value of the kernel.

```
FOR i = 0 to (n-1)
  FOR j = 0 to (n-1)
    dfdx[i] = dWx[i,j] * f[j]   eqn(2.8)
    dfdy[i] = dWy[i,j] * f[j]   eqn(2.8)
  NEXT j
NEXT i
```

Table 2.2: MCI Derivative Calculation Algorithm

Table 2.3 gives an algorithm of how a boundary value problem can be solved with the MCI method. This algorithm is used to solve equations such as eqn(1.43). This linear system of equations has the form of multiplying a matrix ($[K]$) by the solution vector ($\{a\}$) to obtain the forcing vector ($\{f\}$):

$$[K]\{a\} = \{f\} \tag{2.12}$$

Before calculations begin, each particle must be identified as being a member of the field equation or a boundary particle. Boundary particles are then separated for having Dirichlet or Neumann boundary conditions. Particles having Neumann conditions have two values associated with them indicating the normals to the wall in the x and y directions (nx and ny).

```
FOR i in the field equation
  FOR j = 0 to (n-1)
    K[i,j] = ddW[i,j]
  NEXT j
NEXT i
FOR i having Dirichlet conditions
  FOR j = 0 to (n-1)
    K[i,j] = W[i,j]
  NEXT j
NEXT i
FOR i having Neumann conditions
  FOR j = 0 to (n-1)
    K[i,j] = dWx[i,j] * nx[i] + dWy[i,j] * ny[i]
  NEXT j
NEXT i
{a} = INVERSE([K]) * {f}
```

Table 2.3: MCI Partial Differential Equation Solution Algorithm

2.1.4 Conclusions

The Monte Carlo Integration method is very easy to implement in a code. However it suffers from three difficulties:

1. Because values are obtained from averaging the neighbors, derivatives near the boundaries of the material will not be correct. The MCI method inherently requires a symmetric distribution of nodes around each particle to correctly calculate derivatives. When nodes are vacant on a given side of the particle, the derivatives will be poorly calculated. There are various ways of dealing with this problem (Bonet and Kulasegaram, 2002), however the complexity is more severe than claimed.

2. Although values of functions can be obtained by multiplying each particle by a "mass," different sized particles produce values which are not correct. The MCI method works best with many small particles, and larger particles will not be evaluated correctly.

3. Since the method is based off of averaging values from neighbors, there is an inherent viscosity associated with the method. We could reduce the averaging of values by making the value h very small so as to use less particles. In fact, the kernel reproduces the exact value when $h \to 0$. However, when we reduce h to conserve energy in the system, we lose resolution. When we lose resolution, we lose stability, especially in the calculation of derivatives. Hence, there is a contradiction.

2.2 Radial Basis Functions

Instead of using averages from neighbors, it is possible to calculate interpolants by first solving a system of equations for known values. By doing this, we can ensure behavior of the Dirac delta function at each node. Furthermore, by applying differencing operators to the system of equations, we can also calculate derivatives of the interpolants. Problems associated with the geometric distribution of points in the MCI method can be alleviated, however, problems of computation may arise.

2.2.1 Formulation

We say that our continuous function, $\hat{f}$, is a linear combination of a basis function, ϕ, and an unknown weighting value, λ (Cheng et al., 2003).

$$\hat{f}(\underline{x}) = \sum_{\jmath=0}^{n-1} \phi_\imath(r_\jmath)\lambda(\underline{x}_\jmath) \tag{2.13}$$

To perform interpolation, we specify that $\hat{f}(\underline{x} = \underline{x}_\imath) = f(\underline{x}_\imath)$. Using this prescription, we solve eqn(2.13) to find the weights, λ. Because all values of the weighting functions are nodally based, derivatives may pass directly to the basis function.

$$\nabla\hat{f}(\underline{x}) = \sum_{\jmath=0}^{n-1} \nabla\phi_\imath(r_\jmath)\lambda(\underline{x}_\jmath) \tag{2.14}$$

The laplacian is treated in the same way.

$$\nabla^2\hat{f}(\underline{x}) = \sum_{\jmath=0}^{n-1} \nabla^2\phi_\imath(r_\jmath)\lambda(\underline{x}_\jmath) \tag{2.15}$$

Once the weights, λ, are known, derivatives can be found simply by evaluating eqn(2.14).

2.2.2 Prescriptions on basis functions

Two types of basis functions may be chosen. Local basis functions are similar to those used in the MCI techniques. Types of local basis functions include Gaussian curves or inverse multiquadrics. They result in sparse matrices, but have extremely poor convergence on the boundaries. Equation (2.16) shows a Gaussian basis function.

$$\phi\left(r_{\imath\jmath}\right) = e^{-r_{\imath\jmath}^2} \tag{2.16}$$

Global basis functions have the form that distant particles have more influence than neighboring particles. Types of global basis functions include thin plate splines and multiquadrics.

While global basis functions mean that particles have more influence from distant particles than they do from neighboring particles. Equation (2.17) shows a thin plate spline function.

$$\phi\left(r_{ij}\right) = r_{ij}^2 \log r_{ij} \tag{2.17}$$

Multiquadrics can be shown to have exponential convergence (Cheng et al., 2003), however, the matrix which results may be poorly conditioned and very full. Multiquadrics have been found in Lagrangian meshless methods to produce the best calculations. The parameter n involved in multiquadrics affects the type of kernel. If n is small, the kernel becomes a compact, inverse multiquadric. If n is large, the kernel becomes very global. The kernel for multiquadrics is (when $n = 3$):

$$\phi\left(r_{ij}\right) = \left(h^2 + r_x^2 + r_y^2\right)^{n-3/2} \tag{2.18}$$

The gradient is:

$$\nabla_k \phi\left(r_{ij}\right) = \left(2n-3\right) r_k \left(h^2 + r_x^2 + r_y^2\right)^{n-5/2} \tag{2.19}$$

The laplacian is:

$$\nabla^2 \phi\left(r_{ij}\right) = \left(2n-3\right)\left(2h^2 + \left(2n-3\right) r^2\right)\left(h^2 + r_x^2 + r_y^2\right)^{n-7/2} \tag{2.20}$$

2.2.3 Evaluation

We may use Radial Basis Functions to solve a partial differential equation via collocation (Fedoseyev et al., 2002). We will use matrix notation using Einstein summation. For the field equation we have:

$$\nabla^2 \hat{f}_i = \left(\nabla^2 \phi(r_{ij})\right) \lambda_j \tag{2.21}$$

For the flux boundaries, we solve:

$$\nabla f_i = \left(\nabla \phi_{ij}\right) \lambda_j \tag{2.22}$$

For Dirichlet boundaries, we solve:

$$f_i = \phi_{ij} \lambda_j \tag{2.23}$$

Table 2.4 shows the initialization routine for the RBF method. The algorithm is identical to the initiation routine for the MCI method in that the same sort of operators are created.

Table 2.5 shows how derivatives may be calculated using RBFs. What is required first is the weights λ. Afterwards, the differential operators are simple to evaluate.

Table 2.6 gives an algorithm of how a boundary value problem can be solved with Radial Basis Functions. This algorithm, like that for the MCI method, involves solving an equation $[K]\{\lambda\} = \{f\}$.

```
FOR i = 0 to (n-1)
  FOR j = 0 to (n-1)
    CALCULATE r[i,j]        eqn(2.2)
    CALCULATE phi[i,j]      eqn(2.18)
    CALCULATE dphix[i,j]    eqn(2.19)
    CALCULATE dphiy[i,j]    eqn(2.19)
    CALCULATE ddphi[i,j]    eqn(2.20)
  NEXT j
NEXT i
```

Table 2.4: RBF Initialization Algorithm

```
{lambda} = INVERSE([phi]) * {f}          eqn(2.13)
FOR i = 0 to (n-1)
  FOR j = 0 to (n-1)
    dfdx[i] = dphix[i,j] * lambda[j]     eqn(2.14)
    dfdy[i] = dphiy[i,j] * lambda[j]     eqn(2.14)
  NEXT j
NEXT i
```

Table 2.5: RBF Derivative Calculation Algorithm

```
FOR i in the field equation
  FOR j = 0 to (n-1)
    K[i,j] = ddphi[i,j]
  NEXT j
NEXT i
FOR i having Dirichlet conditions
  FOR j = 0 to (n-1)
    K[i,j] = phi[i,j]
  NEXT j
NEXT i
FOR i having Neumann conditions
  FOR j = 0 to (n-1)
    K[i,j] = dphix[i,j] * nx[i] + dphiy[i,j] * ny[i]
  NEXT j
NEXT i
{lambda} = INVERSE([K]) * {f}
```

Table 2.6: RBF Partial Differential Equation Solution Algorithm

2.2.4 Conclusions

Radial Basis Functions suffer difficulties in the form:

1. A local basis will not properly calculate derivatives on the boundaries. This is the same difficulty as found with the MCI method. Therefore, a global basis function must be utilized.

2. Global basis functions suffer from massive ill-conditioning. Conditioning numbers $O(17)$ are common. Furthermore, when particles begin to move and two particles move very close to each other, the matrix will become singular.

3. Global basis functions do not have a good physical interpretation. A particle should have more influence from its neighbor than it does from the furthest particle.

2.3 Multi-dimensional Lagrange Interpolation Polynomials

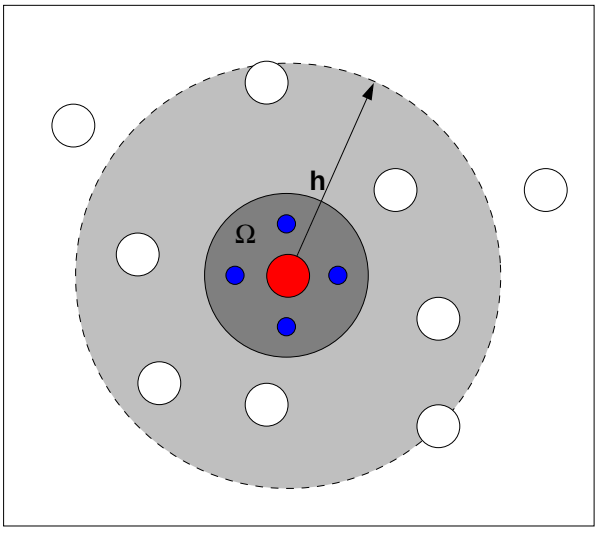
The difficulty with the MCI method is that while we desire great accuracy with $h \to 0$, we also desire stability and resolution when h is very large. Different sized particles can not be used as this will create problems with particle "vacancies." Furthermore, derivatives on the boundaries are poorly calculated. However, what is desirable is that the matrices are very sparse and are not expensive to calculate.

Radial Basis Functions alleviate the accuracy problems with MCI methods. However, we get further problems in that an extra matrix must be inverted for every calculation. Not only is the matrix full, but it usually has a very poor conditioning number. Finally, the physical interpretation of radial basis functions is that distant particles have more influence on a particle than neighbor particles. This can lead a fluid simulation to become unstable.

Instead, a new method is devised using a series of Lagrange polynomial interpolations. The purpose of the MLIP method is to extend the resolution to interpolate for many particles, while still retaining the property of a Dirac delta function. Furthermore, the matrix to be inverted maintains the properties of being sparse and diagonally dominant.

2.3.1 Formulation

Suppose that instead of individual "particles" or "nodes" of fluid, we discretize our domain into actual finite volumes. To represent the specified partition of fluid, we use a set of quadrature points within each volume.



A diagram of integration

- The red particle is center of the integration volume, Ω
- The white particles are neighbor particles
- The length h is the interaction radius
- The blue particles are the positions for Gaussian quadrature

Figure 2.2: MLIP Integration

Our purpose is to relate the deformations of each volume to all others. The Lagrangian interpolation formula expresses a one-dimensional function $g(x)$ as a polynomial $l_n(x)$ of

degree n in a linear equation with a second function $f(x)$ (Hildebrand, 1987).

$$g(x) = l_0(x)f(x_0) + l_1(x)f(x_1) + \cdots + l_n(x)f(x_n) = \sum_{k=0}^{n-1} l_k(x)f(x_k) \tag{2.24}$$

The lagrangian coefficient function $l_\imath(x)$ is a solution of this system of linear equations which also has the property of $l_\imath(x_\jmath) = \delta_{\imath\jmath}$. The coefficient function can be expressed by the monic polynomial:

$$\pi(x) = (x - x_0)(x - x_1)\cdots(x - x_n) \tag{2.25}$$

The solution for the coefficient function becomes:

$$l_\imath(x) = \frac{\pi(x)}{(x - x_\imath)\pi'(x_\imath)} = \prod_{\substack{k=0 \\ k\neq\imath}}^{k-1} \frac{x - x_k}{x_\imath - x_k} \tag{2.26}$$

Changing variables of $g(x) = f(x)$, we obtain:

$$f(x) = \sum_{\jmath=0}^{n-1} l_\jmath(x)f(x_\jmath) + E(x) \tag{2.27}$$

The associated error with this interpolation is:

$$E(x) = \pi(x)\frac{f^{n+1}(\xi)}{(n+1)!} \tag{2.28}$$

If we now desire to calculate the one-dimensional integral, we can express it through a Gaussian quadrature as a function of a weight W

$$\hat{f} = \int_\Omega w(x)f(x)\mathrm{d}x \approx \sum_{\jmath=0}^{n-1} W_\jmath f(x_\jmath) \tag{2.29}$$

The weight W can be evaluated as the weight from a quadrature point $w(x)$ and the Lagrange interpolation:

$$W_\jmath = \sum_{\imath=0}^{n-1} w_\imath(x)l_\imath(x) \tag{2.30}$$

Written in index notation and using Einstein's summation convention,

$$\hat{f} = \int_\Omega w(x)f(x)\mathrm{d}x \approx w_{\imath\jmath}l_\jmath f_\imath \tag{2.31}$$

In one-dimension, the integration occurs within the limits of each particle's size. However, particles "interact" with other particles outside of the integration domain via the interpolating polynomial l.

Taking derivatives of the Lagrange polynomial, we get:

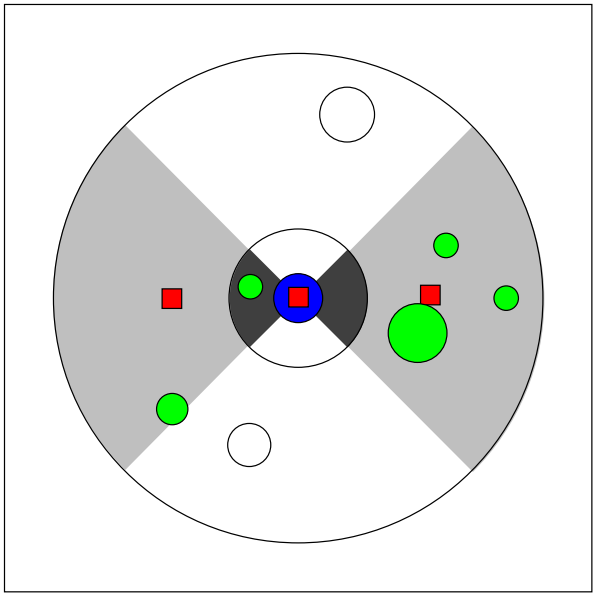
$$\frac{dl_\imath(x)}{dx} = \sum_{\substack{m=1 \\ m\neq\imath}}^{m-1} \frac{1}{x - x_m} \prod_{\substack{k=0 \\ k\neq\imath}}^{k-1} \frac{x - x_k}{x_\imath - x_k} \tag{2.32}$$

$$\frac{d^2l_\imath(x)}{dx^2} = \sum_{\substack{n=1 \\ n\neq\imath}}^{n-1} \frac{1}{x - x_n} \sum_{\substack{m=1 \\ m\neq\imath\neq n}}^{m-1} \frac{1}{x - x_m} \prod_{\substack{k=0 \\ k\neq\imath}}^{n-1} \frac{x - x_k}{x_\imath - x_k} \tag{2.33}$$

2.3.2 Multi-dimensional evaluation

In multiple dimensions, we perform the interpolation for each quadrature point in the particle's volume. Since the Lagrangian polynomial is inherently a one-dimensional function, dimensional "cuts" must be performed. For each quadrature point, in two-dimensions, we divide the surrounding region into two cuts (x and y). We include or exclude points in a cut based on the angle to the point (eqn(2.34)). Each cut uses one-dimensional Lagrangian interpolation.

$$\theta_{\imath\jmath} \equiv \arctan(\underline{x}_\imath - \underline{\xi}_\jmath) \tag{2.34}$$



A diagram for multi-dimensional interpolation

- The blue point is the quadrature point to perform the interpolation
- The green particles are neighboring volumes to be interpolated
- The white particles are excluded from the directional interpolation
- The gray sectors represent the spaces to be interpolated
- The red squares are the calculated "interpolation points"

Figure 2.3: Dimensional-cut for the MLIP method

The interpolating polynomial for each cut passes through every point we give to it. Therefore, if we have a large number of points, the order of the polynomial will become extreme, and the interpolation will become chaotic. Therefore, we will limit the number of

points used in the interpolation to be parabolic. These points will be called "interpolation points."

For each dimensional-cut, we can divide the area into 3 sectors to provide 3 interpolation points. We can calculate the position of the sectors by constructing a histogram for each cut. Each sector from the histogram contains a set of points which influence each center. The location of each "interpolation point" is determined by a weighted average of the particles in each sector.

Table 2.7 shows the algorithm to create a histogram for a dimensional cut. In brief, we create a MCI smoothing technique with a very large smoothing radius. By multiplying the weighting kernel by a penalty vector initially containing values of unity, we can find the most "dense" sector for each cut. After we find the most dense area, we modify the penalty vector to be negative at that location. We repeat the process with the modified penalty vector to find the next most dense sector.

```
function makebins(x,y)
  FOR i = 0 to n
    FOR j = 0 to n
      r[i,j] = radius(x,y)              eqn(2.2)
      w[i,j] = exp(-abs(r[i,j]))
    NEXT j
  penalty(i) = 1.
  NEXT i

  FOR bin = 0 to 2
    location_of_max = max(w*penalty)
    points[bin] = x(location_of_max)
    penalty[location_of_max] = -100
  NEXT bin

  RETURN(points)
```

Table 2.7: MLIP Sectoring Algorithm

Now that we have each cut divided into sectors, we need to interpolate for each "interpolation point" of each sector. We can interpolate for each point by either MCI or RBF. RBFs have been shown to provide the most accurate interpolations within each sector of each cut. By using RBFs for each sector, we obtain the influence for each point by inverting a very small (and not ill-conditioned) RBF interpolation matrix. Table 2.8 shows the algorithm for performing a local sector interpolation. The inputs for the function are a matrix of the distances, r for each particle in a sector to each other, and a vector of the distance r_{center} from each particle to the "interpolation point" of the sector.

The procedure for this method is best performed in a particle-by-particle basis. For each particle, we call an initialization algorithm and then multiply the resulting Lagrange

```
function weight(r,r_center)
  FOR i = 0 to n
    FOR j = 0 to n
      weight[i,j] = phi(r[i,j])   eqn(2.18)
    NEXT j
    wbin[i] = phi(r_center[i])    eqn(2.18)
  NEXT i

  w = wbin * INVERSE(weight)

  RETURN(w)
```

Table 2.8: MLIP Local Sector Interpolation Algorithm

polynomial vectors by the sector weight. This algorithm for each quadrature point i is shown in table 2.9.

```
FOR j = 0 to (n-1)
  CALCULATE r[j]                              eqn(2.2)
  CALCULATE theta[j]                          eqn(2.34)
NEXT j

FOR k = 0 to dim
  CREATE points[k]                            algorithm(2.7)
  FOR p = 0 to 2
    CALCULATE w[j,p]                          algorithm(2.8)
  NEXT p
  l += Lagrange(points[k]) * w[j,p]           eqn(2.26)
  dl[k] = dLagrange(points[k]) * w[j,p]       eqn(2.32)
  ddl += ddLagrange(points[k]) * w[j,p]       eqn(2.33)
NEXT k
```

Table 2.9: MLIP Initialization Algorithm

Besides the initiation routine, we can solve an equation or calculate derivatives in the same manner as with the MCI method using algorithms shown in Tables 2.2 and 2.3.

2.3.3 Conclusions

1. The MLIP method has the desirable property of having a relatively large interaction radius with other particles, but preserves the Dirac delta property.

2. The matrices created using the MLIP method are sparse and diagonally dominant.

This is extremely beneficial for machine storage and the expense to invert.

3. The use of directional cuts and choosing interpolating points has the potential of producing spurious and incorrect results. However, by using quadrature points for each particle, the effect of any incorrect results can be seriously reduced.

4. There is a considerable amount of overhead in performing the initialization routine. However, by using a language such as C++, the sparsity of the method can be utilized in a particle-by-particle manner through the creation of a particle class.

Chapter 3

Results

To evaluate meshless Lagrangian methods, there are two components which must be tested. The first component to be evaluated is the accuracy of the computational methods. We will evaluate the accuracy of the Monte Carlo Integration, Radial Basis Functions, and Multi-dimensional Lagrange Interpolating Polynomials. For the same type of function, the computational methods were tested on different types of particle arrangements.

The second component of the meshless Lagrangian methods to be evaluated is the accuracy of the fluid dynamic models. The equations of motion were applied to different types of simulations. We will evaluate the properties of the Moving Particle Semi-implicit equations as well as the Lagrange Implicit Fraction Step equations.

3.1 Methods of Computation

There were three different particle distributions used for testing methods of computation:

- Full structured grid
- Sparse structured grid
- Unstructured grid

The simulation involved using a function f from eqn(3.2) and creating an interpolation, gradients, and Laplacian operators for each method. The function from eqn(3.2) is set up to be similar to the pressure from a breaking dam problem. It is a solution of a boundary value problem (eqn(3.1)) with a domain (Ω): $x = (1, x_{max})$ and $y = (1, y_{max})$. The free surface is

considered to be on $x = x_{max}$ and $y = y_{max}$. The walls are on $x = 1$ and $y = 1$.

$$\boxed{\begin{aligned} &1)\ \nabla^2 f = 0 \text{ in } \Omega \\ &2)\ f = 0 \text{ on } \Gamma_d, \text{ the free surface} \\ &3)\ \frac{\partial f}{\partial x} = 0 \text{ on } x = 1 \\ &4)\ \frac{\partial f}{\partial y} = -64 \text{ on } y = 1 \end{aligned}} \tag{3.1}$$

The solution to this boundary value problem is our function f:

$$f = \frac{1}{4}\left(a\left(x-1\right)^2 - a\left(x_{max}\right)^2\right)\left(128\left(y-1\right) - a\left(y-1\right)^2 - 128y_{max} + a\left(y_{max}\right)^2\right) \tag{3.2}$$

with the parameter a given as a function of the domain:

$$a = 16\frac{4x_{max}^2 - \sqrt{16x_{max}^4 - x_{max}^2 y_{max}}}{x_{max}^2 y_{max}} \tag{3.3}$$

Figures 3.1-3.4 show the function f as well as the analytical derivatives.

The discrete interpolation was conducted for all three methods of computation.

- Monte Carlo Integration
- Radial Basis Functions
- Multi-dimensional Lagrange Interpolating Polynomials

Error estimates for an analytical function g and an interpolated value $\hat{g}$ are given by:

$$\text{error} = \frac{\sum_{\imath=0}^{n-1} |g(\underline{x} = \underline{x}_\imath) - \hat{g}(\underline{x}_\imath)|}{\sum_{\imath=0}^{n-1} |g(\underline{x} = \underline{x}_\imath)|} \tag{3.4}$$

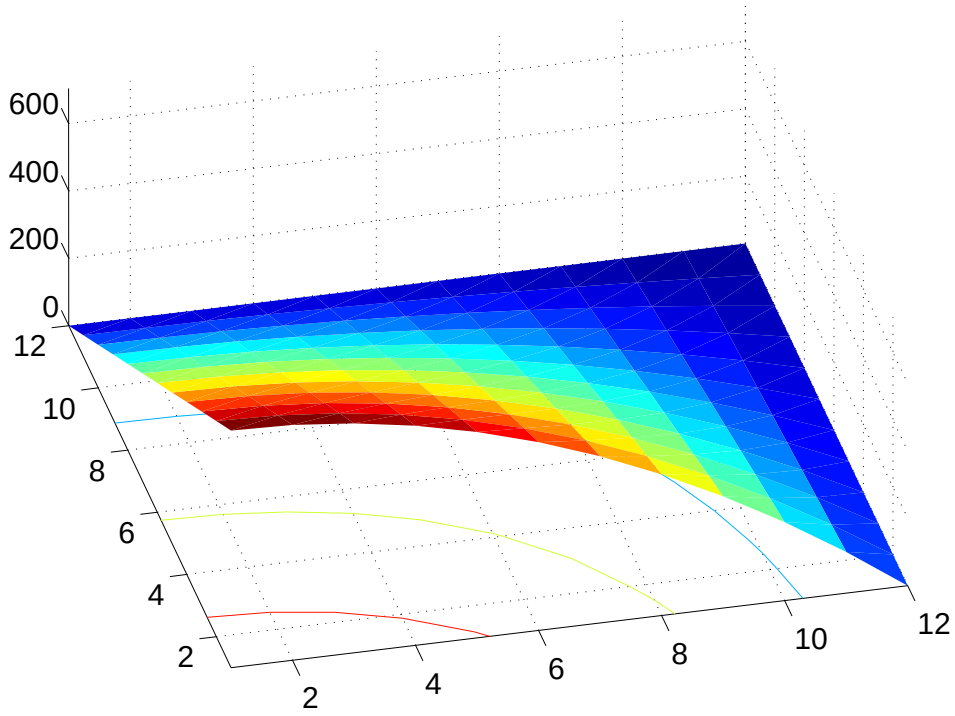


Figure 3.1: Function f of eqn(3.2)

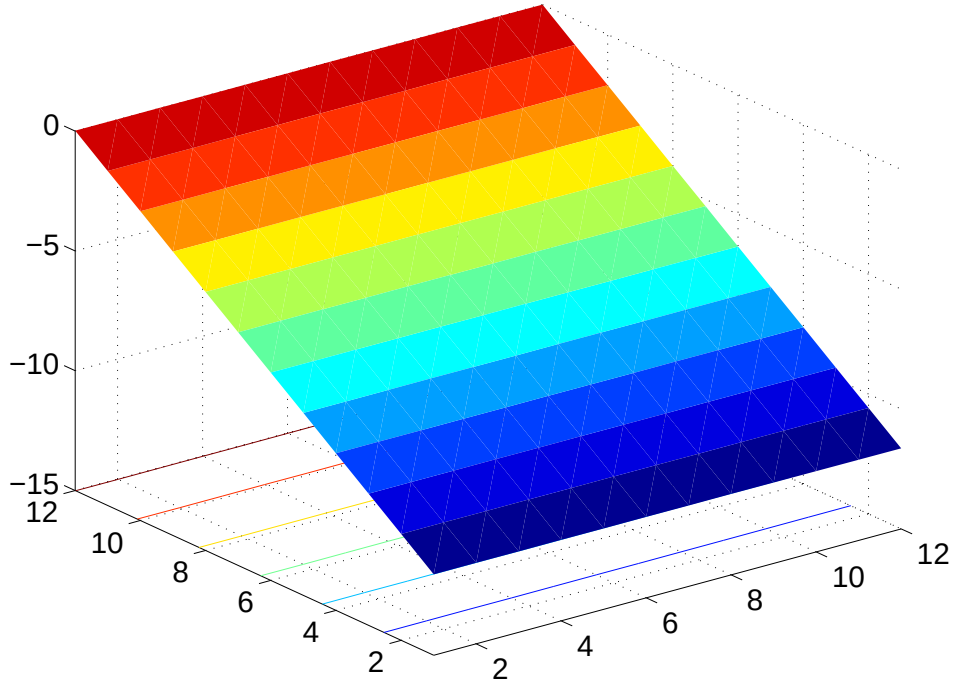


Figure 3.2: $\nabla^2 f$ of eqn(3.2)

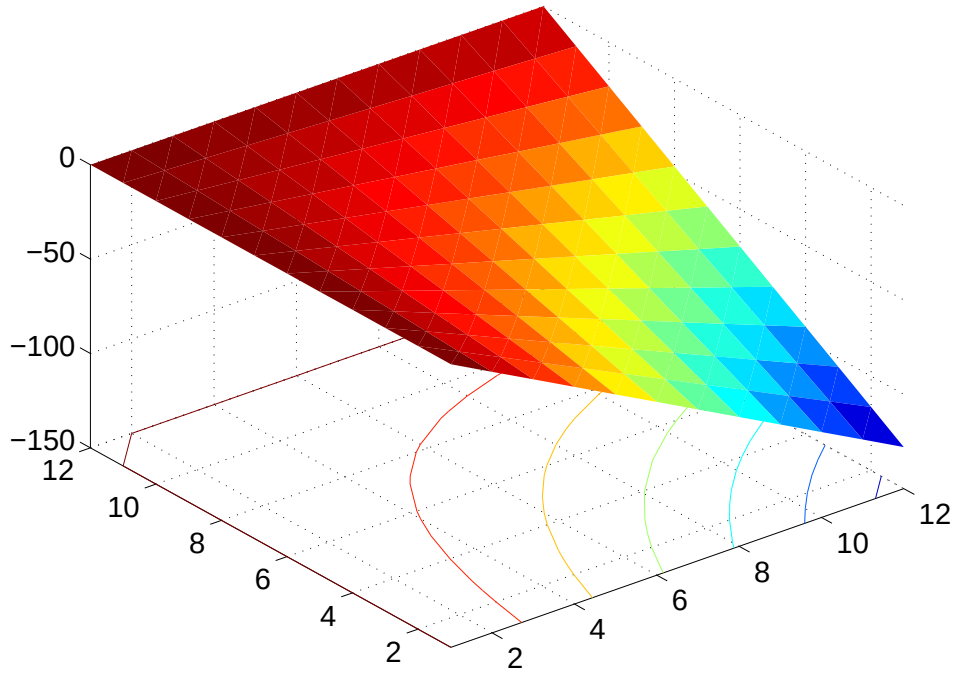


Figure 3.3: $\frac{\partial f}{\partial x}$ of eqn(3.2)

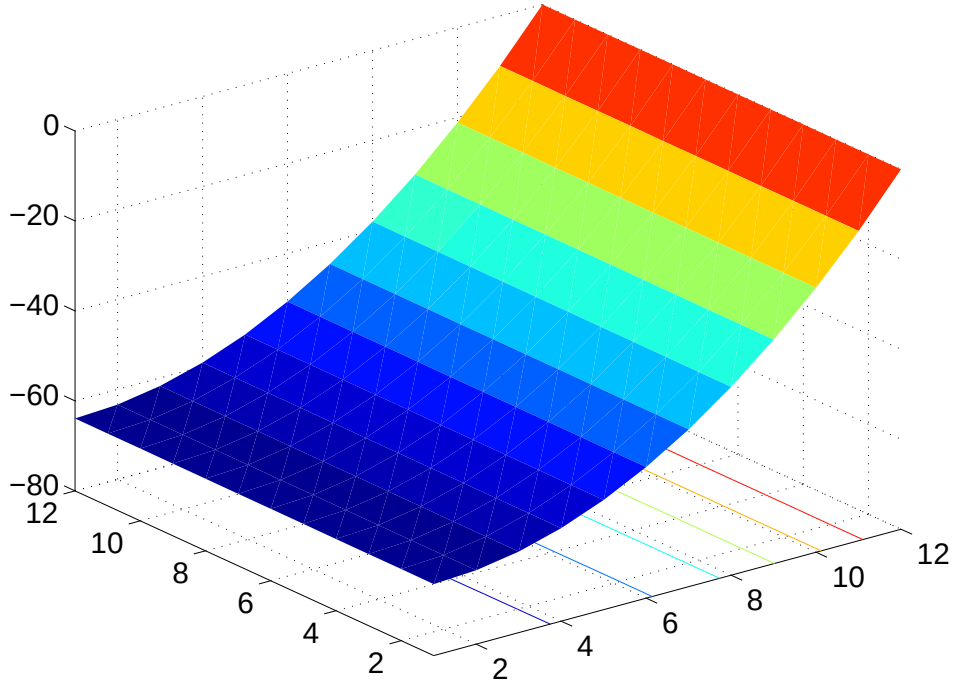


Figure 3.4: $\frac{\partial f}{\partial y}$ of eqn(3.2)

3.1.1 Full structured grid

The structured grid involves 144 particles. It has a range from 1 to 12 in both x and y axes. Figure 3.5 shows the arrangement of particles.

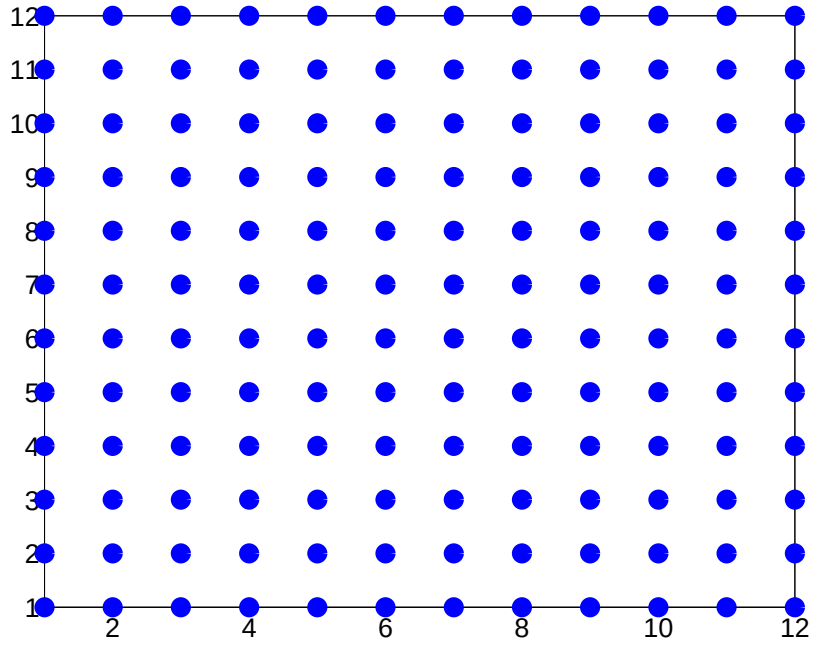


Figure 3.5: Full structured grid

For all results shown below, the analytical values are shown by a colored surface. The discrete values shown numerically are represented by blue circles. When a mesh for the interpolated values can be created, these values are linked together by a transparent surface.

A) Monte Carlo Integration

Figure 3.6 shows the MCI interpolation on a structured grid as obtained by eqn(2.9). Notice that there is a small amount of error between the true value and the interpolated value. This is due to an averaging between nodes. Figure 3.7 shows the Laplacian interpolation on a structured grid from eqn(2.11). There is a considerable amount of error at the boundaries. However, the error in the center of the domain is almost negligible.

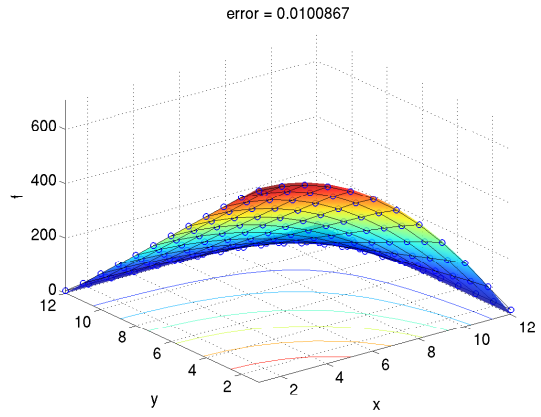


Figure 3.6: MCI: $\hat{f}$ on a full structured grid

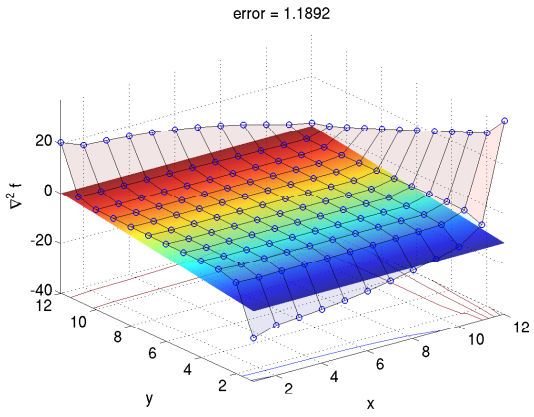


Figure 3.7: MCI: $\nabla^2 \hat{f}$ on a full structured grid

Gradients are obtained from algorithm(2.2). Figure 3.8 shows the gradient in the x direction on a structured grid. There is an extreme amount of error on $x = 1$. Figure 3.9 shows the gradient in the y direction on a structured grid. The same extreme error is also present on $y = 1$.

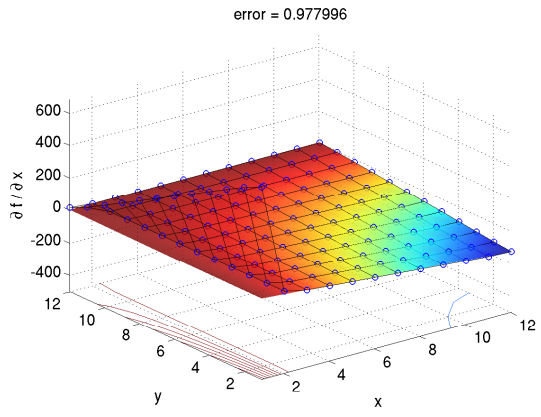


Figure 3.8: MCI: $\frac{\partial \hat{f}}{\partial x}$ on a full structured grid

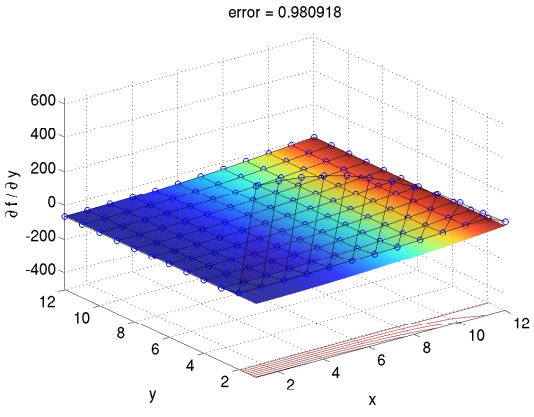


Figure 3.9: MCI: $\frac{\partial \hat{f}}{\partial y}$ on a full structured grid

In terms of the derivatives in the MCI method, boundary points suffer from accuracy because of a vacancy of points on a given side. This error is increased when the value of the gradient is very large.

B) Radial Basis Functions

Figure 3.10 shows the RBF interpolation on a full structured grid as obtained by eqn(2.13). There is no error for the interpolation since the values are solved for via Gaussian - elimination. Figure 3.11 shows the Laplacian interpolation on a structured grid. There is a small amount of error on the boundaries, but it is still very good.

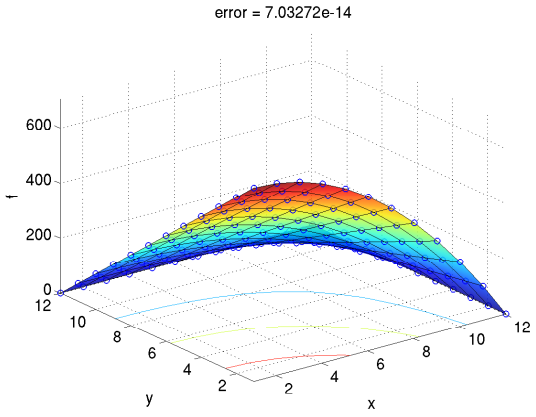


Figure 3.10: RBF: $\hat{f}$ on a full structured grid

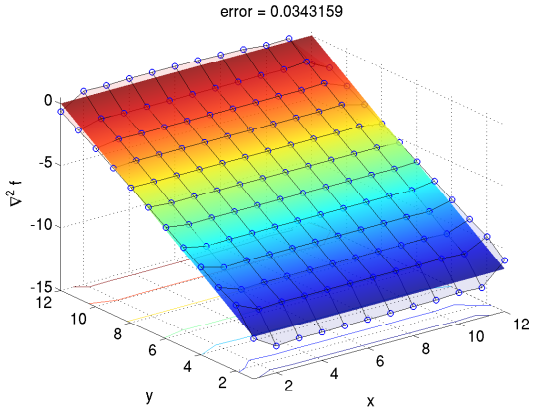


Figure 3.11: RBF: $\nabla^2 \hat{f}$ on a full structured grid

Figures 3.12 and 3.13 show the gradients in the x and y directions. There is almost no error on these grids.

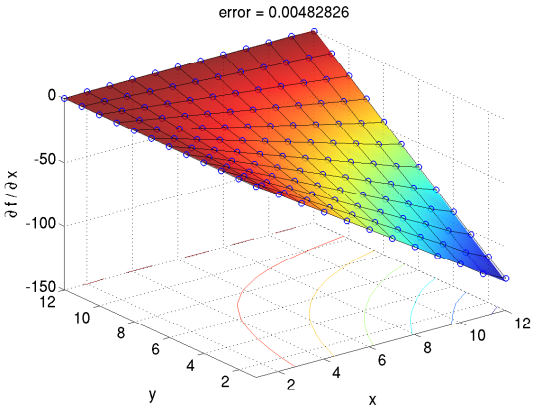


Figure 3.12: RBF: $\frac{\partial \hat{f}}{\partial x}$ on a full structured grid

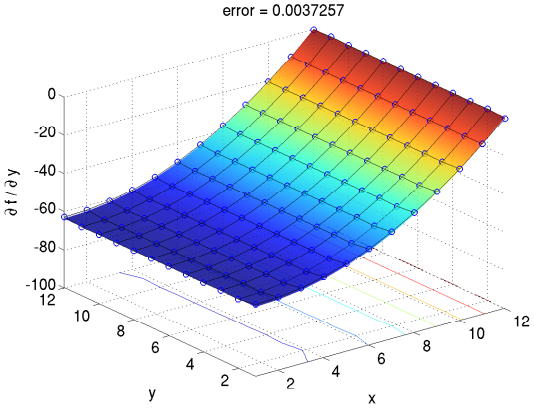


Figure 3.13: RBF: $\frac{\partial \hat{f}}{\partial y}$ on a full structured grid

C) Multi-dimensional Lagrange Interpolating Polynomials

Figure 3.14 shows the interpolation from the MLIP method. The matrix generated by the function on this structured grid is the identity matrix. Figure 3.15 shows the Laplacian. There is very little error in the Laplacian.

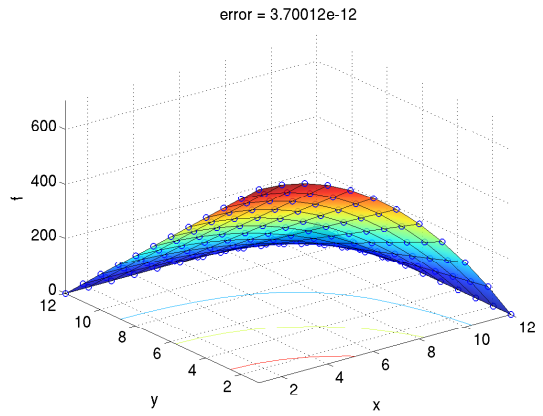


Figure 3.14: MLIP: $\hat{f}$ on a full structured grid

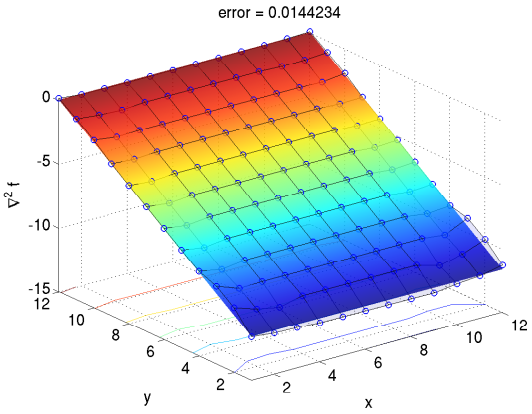


Figure 3.15: MLIP: $\nabla^2 \hat{f}$ on a full structured grid

Figures 3.16 and 3.17 show the gradients in the x and y directions. There is almost no error on these grids.

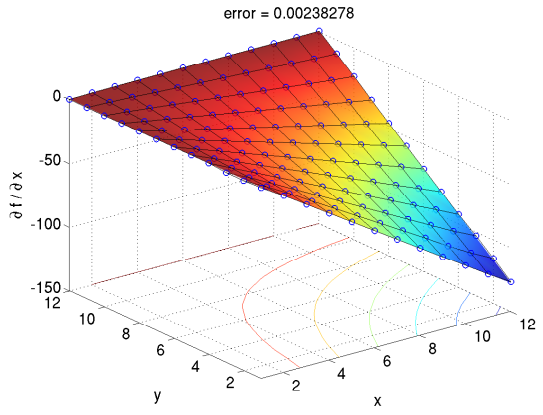


Figure 3.16: MLIP: $\frac{\partial \hat{f}}{\partial x}$ on a full structured grid

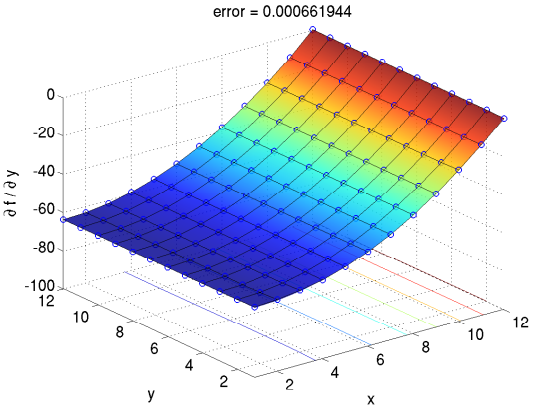


Figure 3.17: MLIP: $\frac{\partial \hat{f}}{\partial y}$ on a full structured grid

3.1.2 Structured sparse grid

The sparse grid is generated via a quadtree method (Wang et al., 1999). The quadtree method is set up to search for particles near boundaries. Particles near boundaries are subdivided until a suitable mesh division is created.

The quadtree method has very nice characteristics of placing more particles where needed. Furthermore, where a large number of particles are unnecessary, these areas are left to only a few particles having very large masses. There were only 70 particles used in the sparse grid.

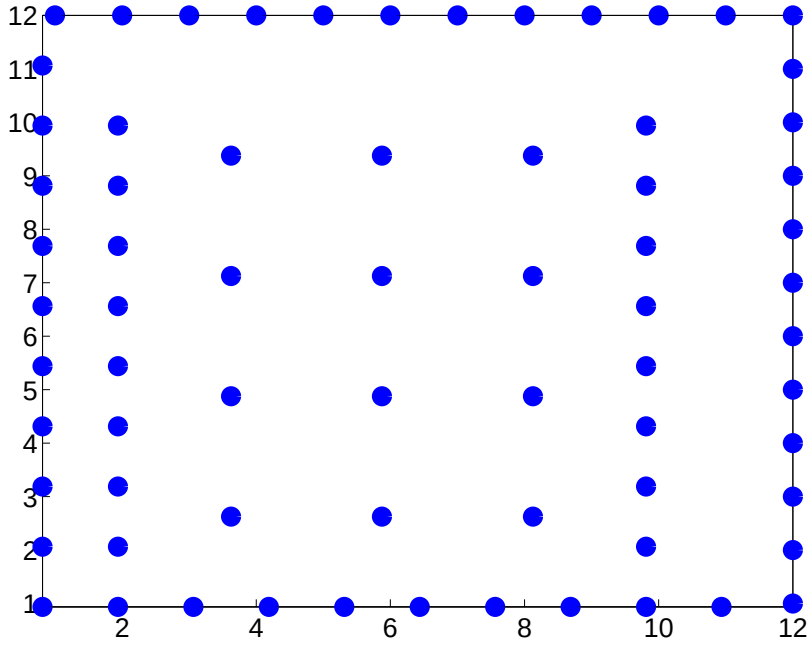


Figure 3.18: Sparse structured grid

A) Monte Carlo Integration

Figure 3.19 shows the MCI interpolation on a sparse grid. Because of the large separation between points, the interpolation actually improves on this grid due to less averaging. Figure 3.20 shows the Laplacian interpolation on a sparse grid. In calculating derivatives, the averaging of method breaks down for "large particles." There is a tremendous amount of error, and the method is entirely unusable for this grid arrangement.

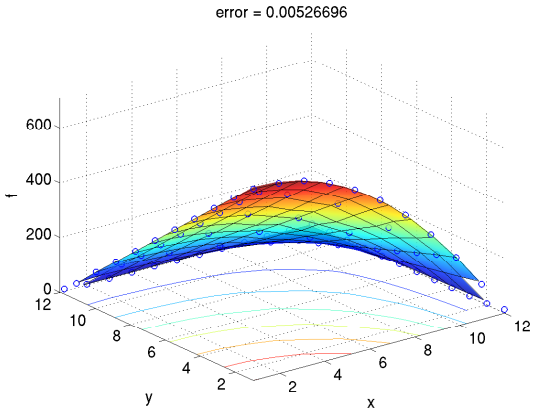


Figure 3.19: MCI: $\hat{f}$ on a sparse structured grid

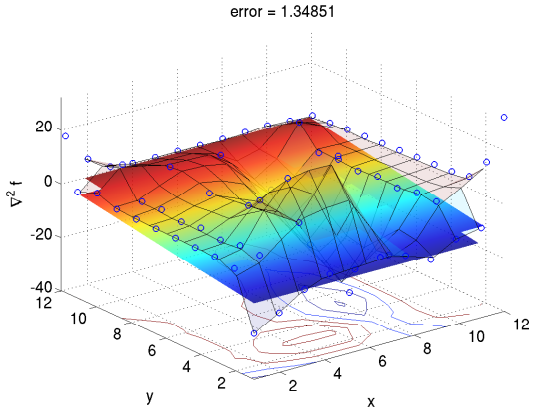


Figure 3.20: MCI: $\nabla^2 \hat{f}$ on a sparse structured grid

Figures 3.21 and 3.22 show the gradients. Gradient operations in the MCI method perform poorly between differently sized particles.

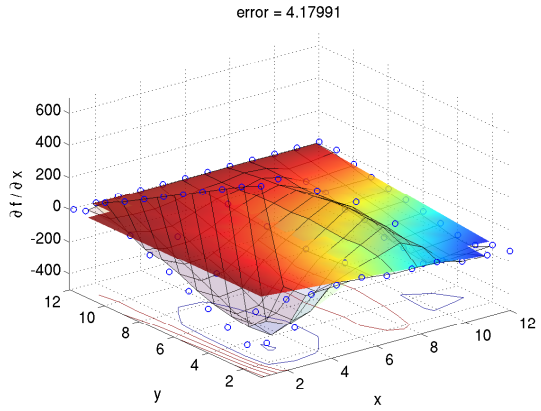


Figure 3.21: MCI: $\frac{\partial \hat{f}}{\partial x}$ on a sparse structured grid

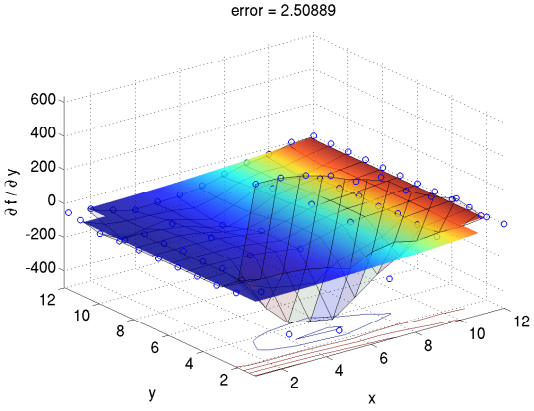


Figure 3.22: MCI: $\frac{\partial \hat{f}}{\partial y}$ on a sparse structured grid

B) Radial Basis Functions

Figure 3.23 shows the RBF interpolation on a sparse grid. Problems of "large particles" associated with the MCI method are fully alleviated. However, because of the large differences in the sizes of particles, the eigenvalues of the matrix become more separated, leading to a larger conditioning number in the matrix to be inverted from algorithm(2.6). The conditioning number to solve the boundary value problem is $3(10)^9$. Figure 3.24 shows the Laplacian interpolation on a sparse grid. The error on the boundaries remains, but is still acceptable.

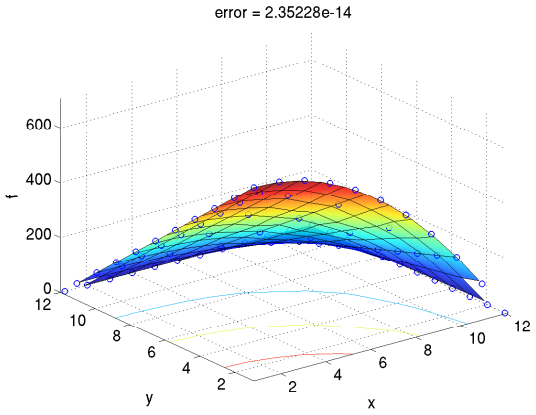


Figure 3.23: RBF: $\hat{f}$ on a sparse structured grid

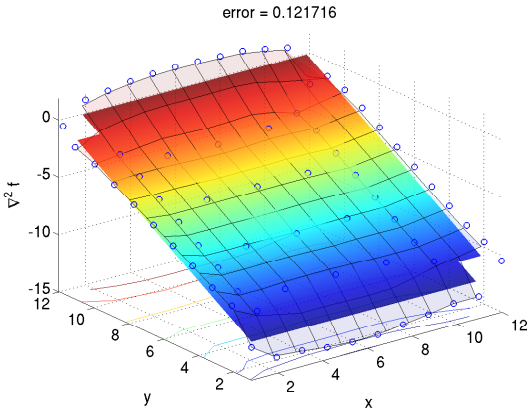


Figure 3.24: RBF: $\hat{f}$ on a sparse structured grid

Figures 3.25 and 3.26 show the gradients in the x and y directions. There is almost no error on these grids.

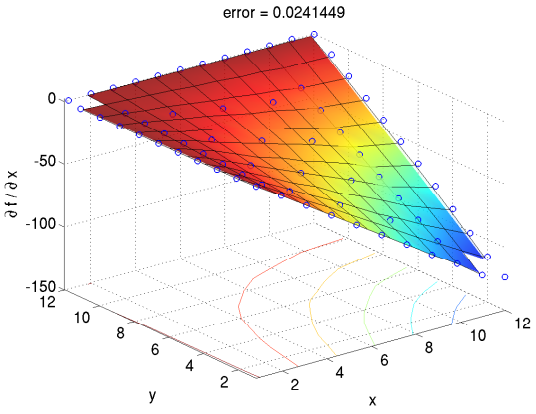


Figure 3.25: RBF: $\frac{\partial \hat{f}}{\partial x}$ on a sparse structured grid

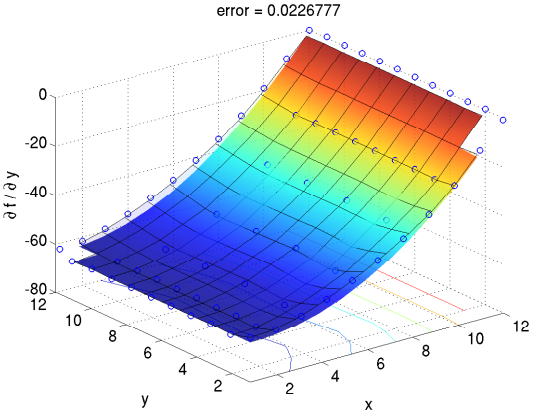


Figure 3.26: RBF: $\frac{\partial \hat{f}}{\partial x}$ on a sparse structured grid

C) Multi-dimensional Lagrange Interpolating Polynomials

The MLIP method works well on a sparse grid work well while also alleviating the problems of the MCI method. Furthermore, the eigenvalues in the resulting matrix do not have the same problems as associated with Radial Basis Functions. The conditioning number to solve the boundary value problem is only 214. Figure 3.28 shows the Laplacian interpolation on a sparse grid. There is even less error than with the Radial Basis Functions, but small errors remain on the boundaries.

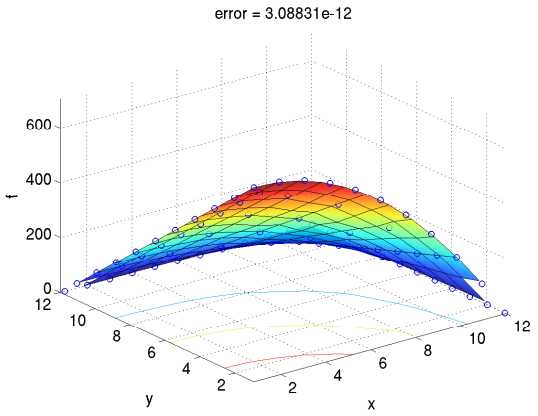


Figure 3.27: MLIP: $\hat{f}$ on a sparse structured grid

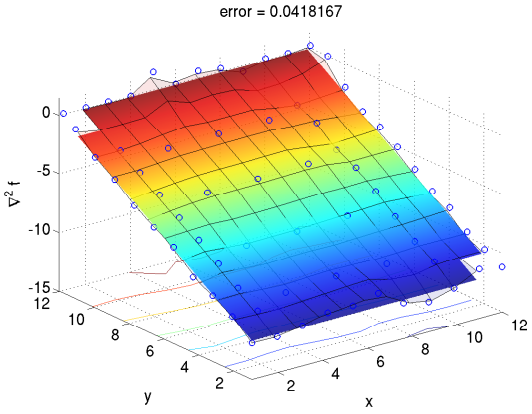


Figure 3.28: MLIP: $\nabla^2 \hat{f}$ on a sparse structured grid

Figures 3.29 and 3.30 show the gradients in the x and y directions. There is almost no error in the calculation of gradients.

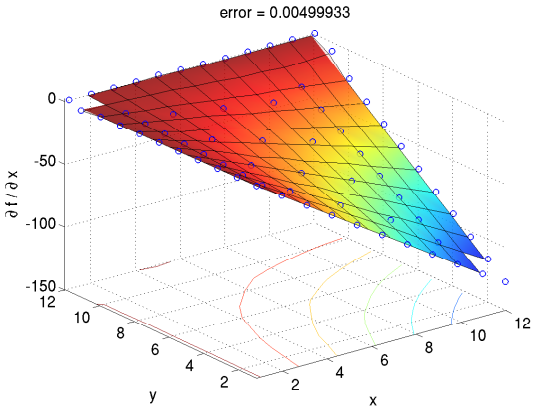


Figure 3.29: MLIP: $\frac{\partial \hat{f}}{\partial x}$ on a sparse structured grid

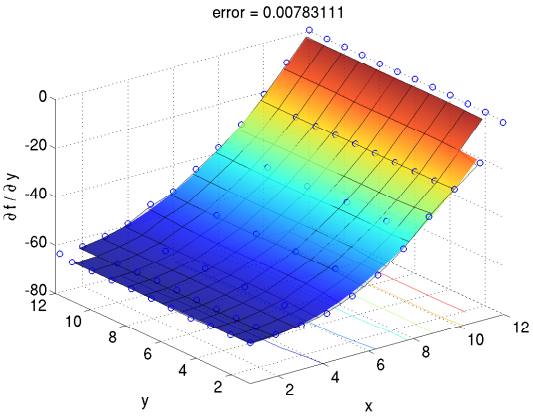


Figure 3.30: MLIP: $\frac{\partial \hat{f}}{\partial y}$ on a sparse structured grid

3.1.3 Unstructured grid

This unstructured grid is set to simulate action after particles have been moved slightly relative to an initially structured grid as in figure 3.5. Some particles move very close to other particles, while other particles move further apart.

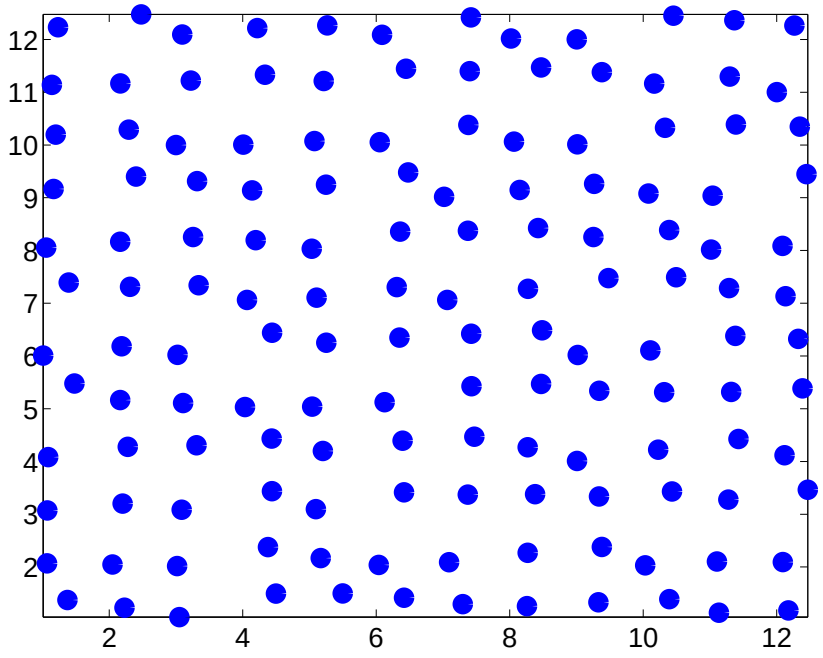


Figure 3.31: Unstructured grid

A) Monte Carlo Integration

Figure 3.32 shows the MCI interpolation on a structured grid. The error present in the structured grid is the same error in the unstructured grid. However, the amount of error does not change dramatically after particles have been moved.

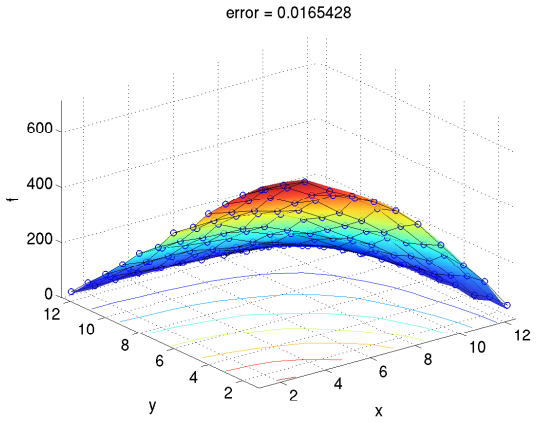


Figure 3.32: MCI: $\hat{f}$ on an unstructured grid

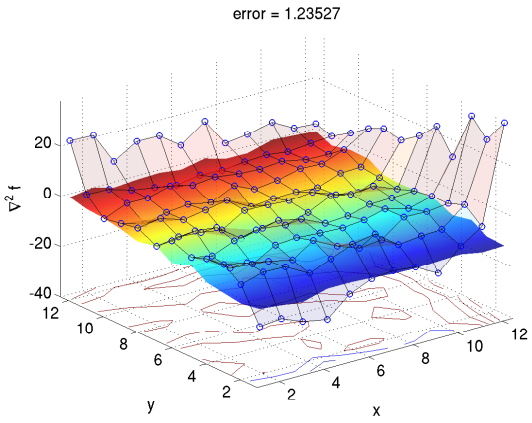


Figure 3.33: MCI: $\nabla^2 \hat{f}$ on an unstructured grid

Figures 3.34 and 3.35 show the gradients in the x and y directions. The error on the boundaries is still present, however, the error in the middle of the mesh changes only slightly.

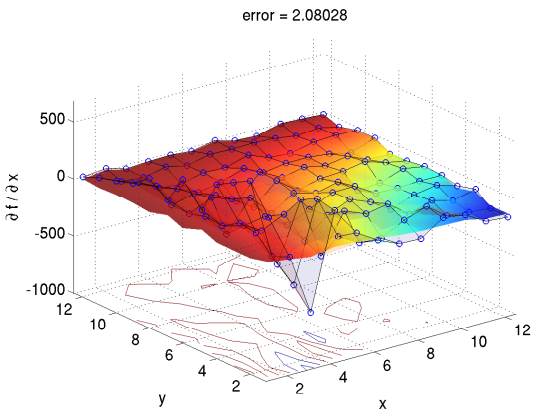


Figure 3.34: MCI: $\frac{\partial \hat{f}}{\partial x}$ on an unstructured grid

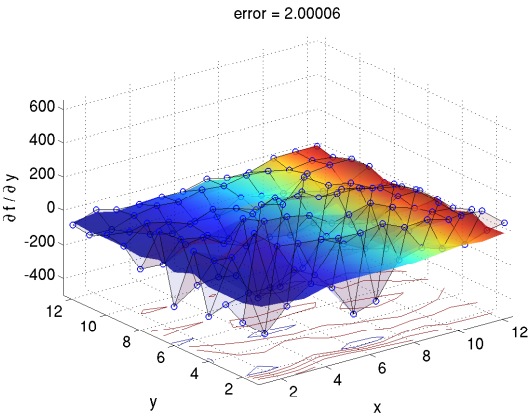


Figure 3.35: MCI: $\frac{\partial \hat{f}}{\partial y}$ on an unstructured grid

B) Radial Basis Functions

Figure 3.36 shows the RBF interpolation on a disordered grid. There is no error in this interpolation. However, figure 3.37 shows the Laplacian interpolation on a disordered grid. Because certain particles are too close to one-another, the calculations can break down and produce very spurious results. When applied to a fluid code, this can cause the simulation to break down.

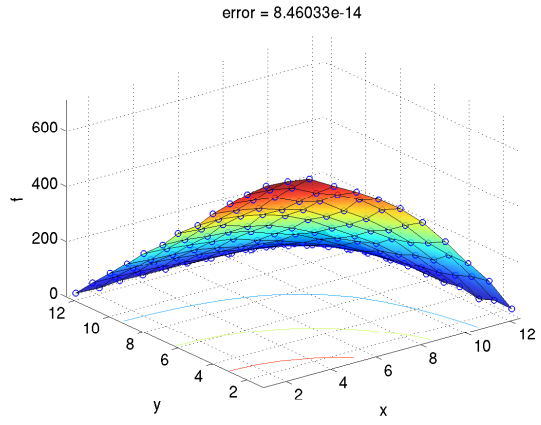


Figure 3.36: RBF: $\hat{f}$ on an unstructured grid

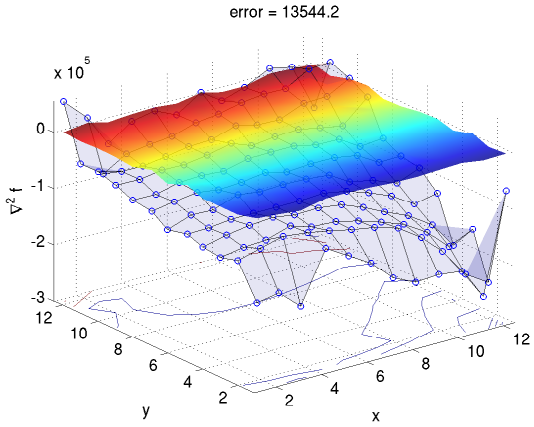


Figure 3.37: RBF: $\nabla^2 \hat{f}$ on an unstructured grid

Figures 3.38 and 3.39 show the gradients in the x and y directions. The calculation of gradients is still very good.

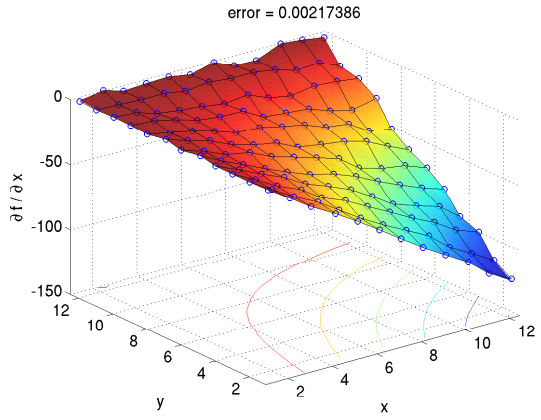


Figure 3.38: RBF: $\frac{\partial \hat{f}}{\partial x}$ on an unstructured grid

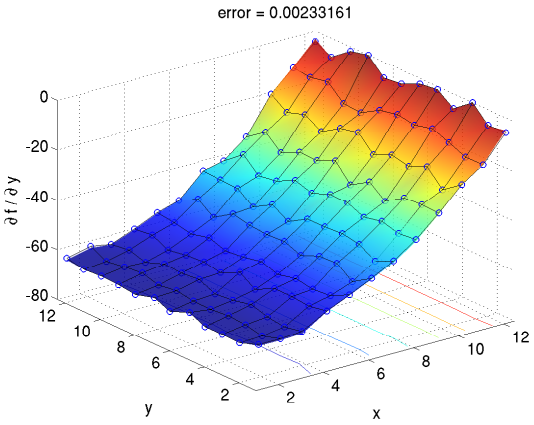


Figure 3.39: RBF: $\frac{\partial \hat{f}}{\partial y}$ on an unstructured grid

C) Multi-dimensional Lagrange Interpolating Polynomials

The interpolation, figure 3.40, is shown to be exact, while the Laplacian, figure 3.41 shows the smallest degree of error from any method.

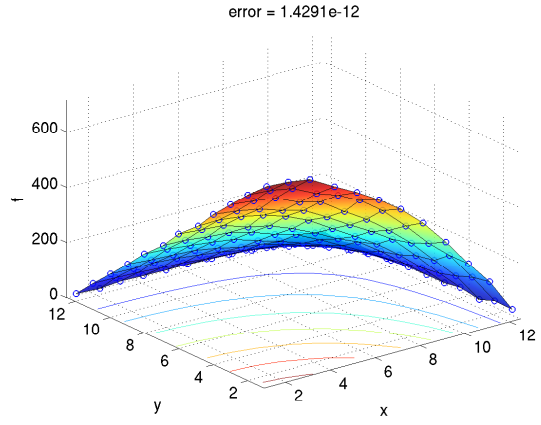


Figure 3.40: MLIP: $\hat{f}$ on an unstructured grid

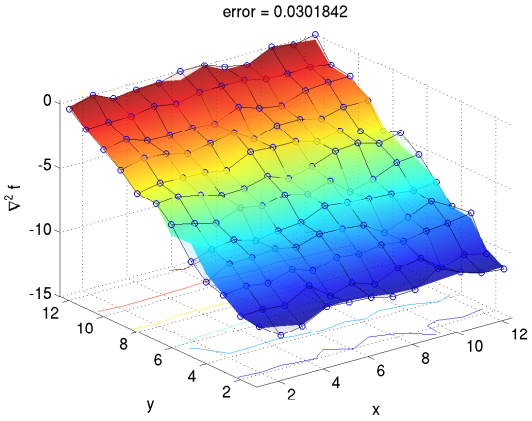


Figure 3.41: MLIP: $\nabla^2 \hat{f}$ on an unstructured grid

The gradients, shown in figures 3.42 and 3.43 show excellent accuracy for a disordered mesh.

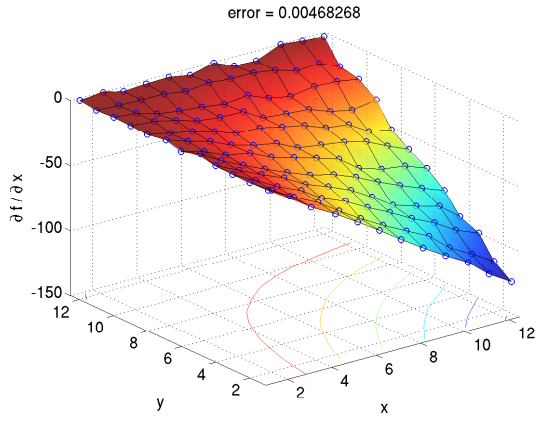


Figure 3.42: MLIP: $\frac{\partial \hat{f}}{\partial x}$ on an unstructured grid

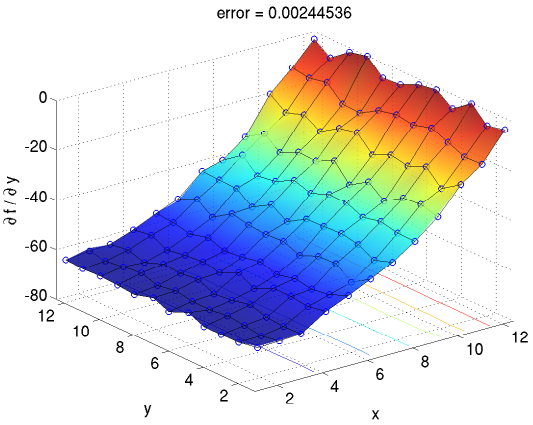


Figure 3.43: MLIP: $\frac{\partial \hat{f}}{\partial y}$ on an unstructured grid

3.2 Fluid Codes

For a meshless Lagrangian method, we may use a combination of any equation of motion and method of computation. In total, there are nine different combinations we can use, shown in table 3.1.

Equations of Motion	Method of Computation
Spherical Particle Hydrodynamics (SPH)	Monte Carlo Integration (MCI)
Moving Particle Semi-implicit (MPS)	Radial Basis Functions (RBF)
Lagrange Implicit Fraction Step (LIFS)	Multi-d Lagrange Interpolating Polynomials (MLIP)

Table 3.1: Equations of Motion and Methods of Computations

The codes chosen for testing some of the theories included:

A MPS and Monte Carlo Integration (MPS - MCI)

B Lagrange Implicit Fraction Step and Radial Basis Functions (LIFS - RBF)

The SPH equations were not investigated because a large amount of research has already been conducted on the subject. Instead, it was decided to investigate implicit codes which did not use an equation of state to calculate the pressure. Multi-dimensional Lagrange Interpolating Polynomials also were not used because an efficient C++ algorithm has not been written yet.

The MPS - MCI code was written in C++. The code made use of the sparsity of the MCI method, and its evaluation was very quick. The LIFS - RBF code was written in Matlab. Because of Matlab's robust matrix inversion, it was utilized since the matrices involved in Radial Basis Functions are very full and ill-conditioned.

Two types of experiments were performed using both methods. Each experiment was used to evaluate the accuracy and robustness of the codes. To simplify the problem as well as to investigate the accuracy of the simulations, viscosity was set to zero. This was done to evaluate if there was any numerical damping involved in the computations.

3.2.1 Dam Break

The dam break involves a water column of height H and length x_0 in which one side of the column is released at time $t = 0^+$. This simulation allowed for the evaluation of boundary conditions as well as a comparison to experimental data.

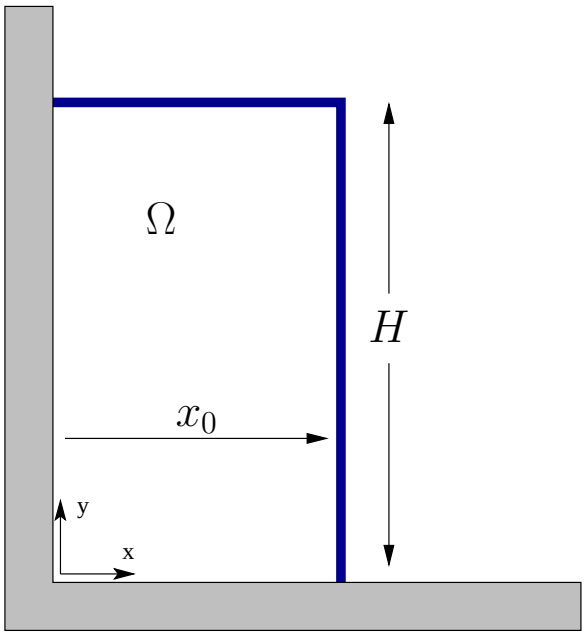


Figure 3.44: Diagram of Dam Break Experiment

A) Moving Particle Semi-Implicit and Monte Carlo Integration

The propagation of the water can be compared to published results (Shao and Lo, 2003), shown in figure 3.45. English units were used in the simulation with $g = 32.2 ft/s^2$ and $\rho = 1.99 slugs/ft^3$.

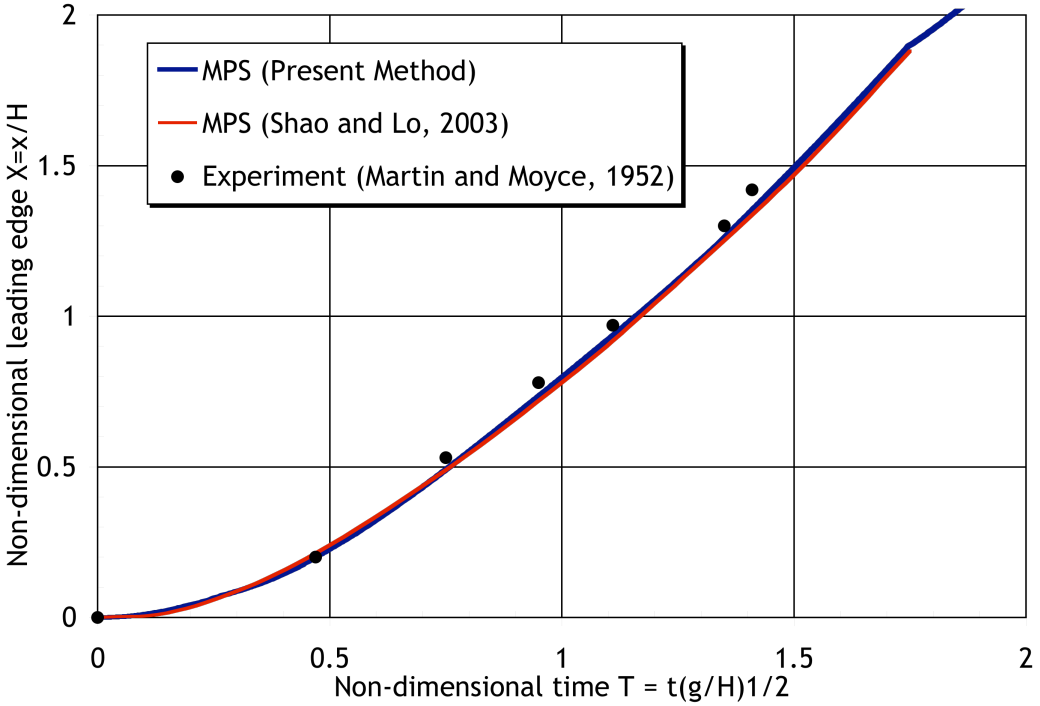


Figure 3.45: Comparison of dam breaks

Figure 3.46 shows the propagation of the water as it decays from its initial position. Identification of the free surface particles utilized inherent defects in Monte Carlo Integration. Particle vacancies were identified by a lack of symmetry in the derivative matrices, dwx and dwy. If the lack of symmetry was over a given limit, the particle would be tagged as being on the free surface.

Figure 3.47 shows the water column hitting another wall on the opposite side. As the water descended from the opposite wall, a small vortex was created in the bottom right corner of the fluid. The presence of this vorticity indicated that there was numerical damping in the Monte Carlo integration technique.

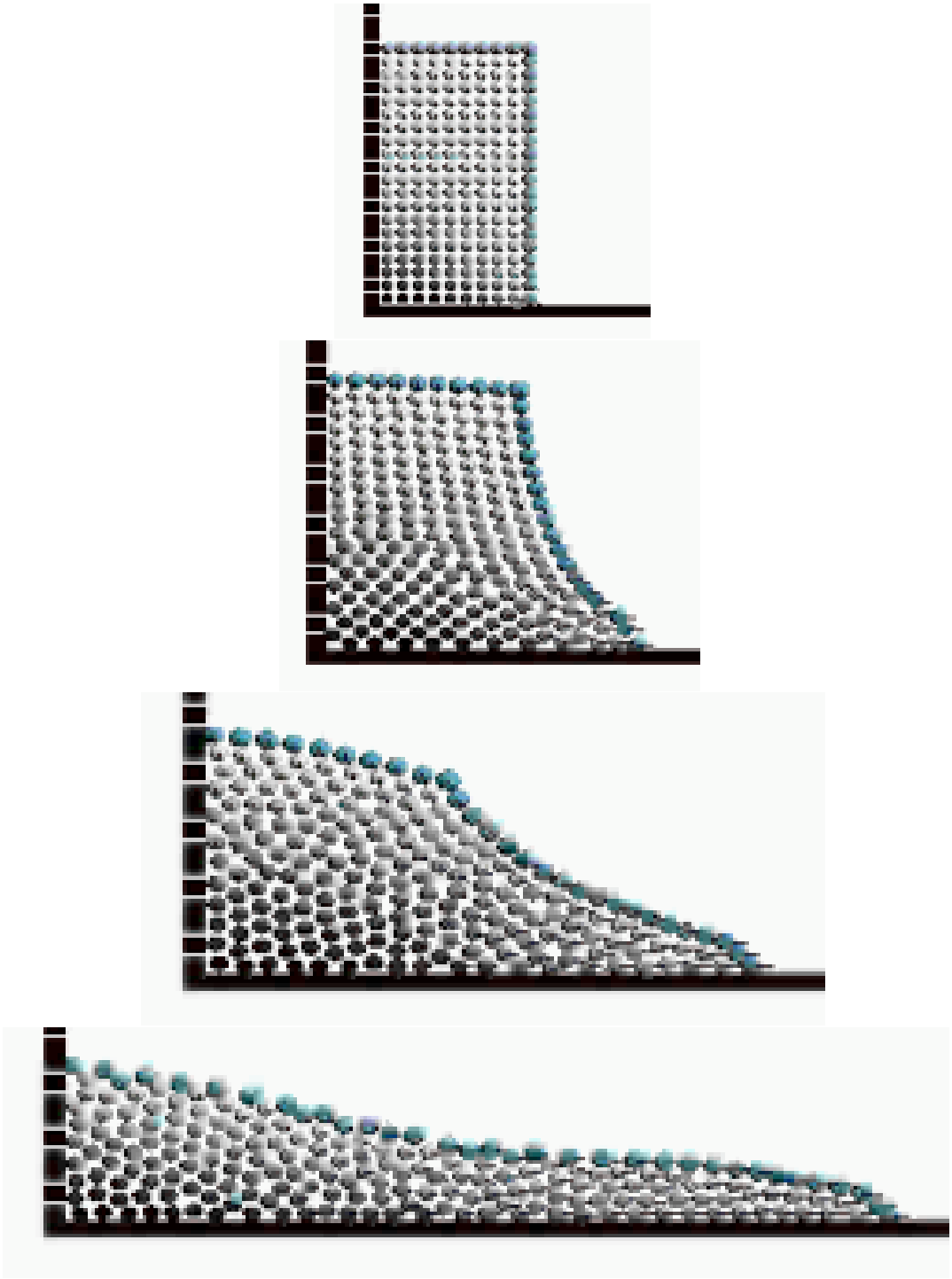


Figure 3.46: Initial decay of water column using MPS - MCI

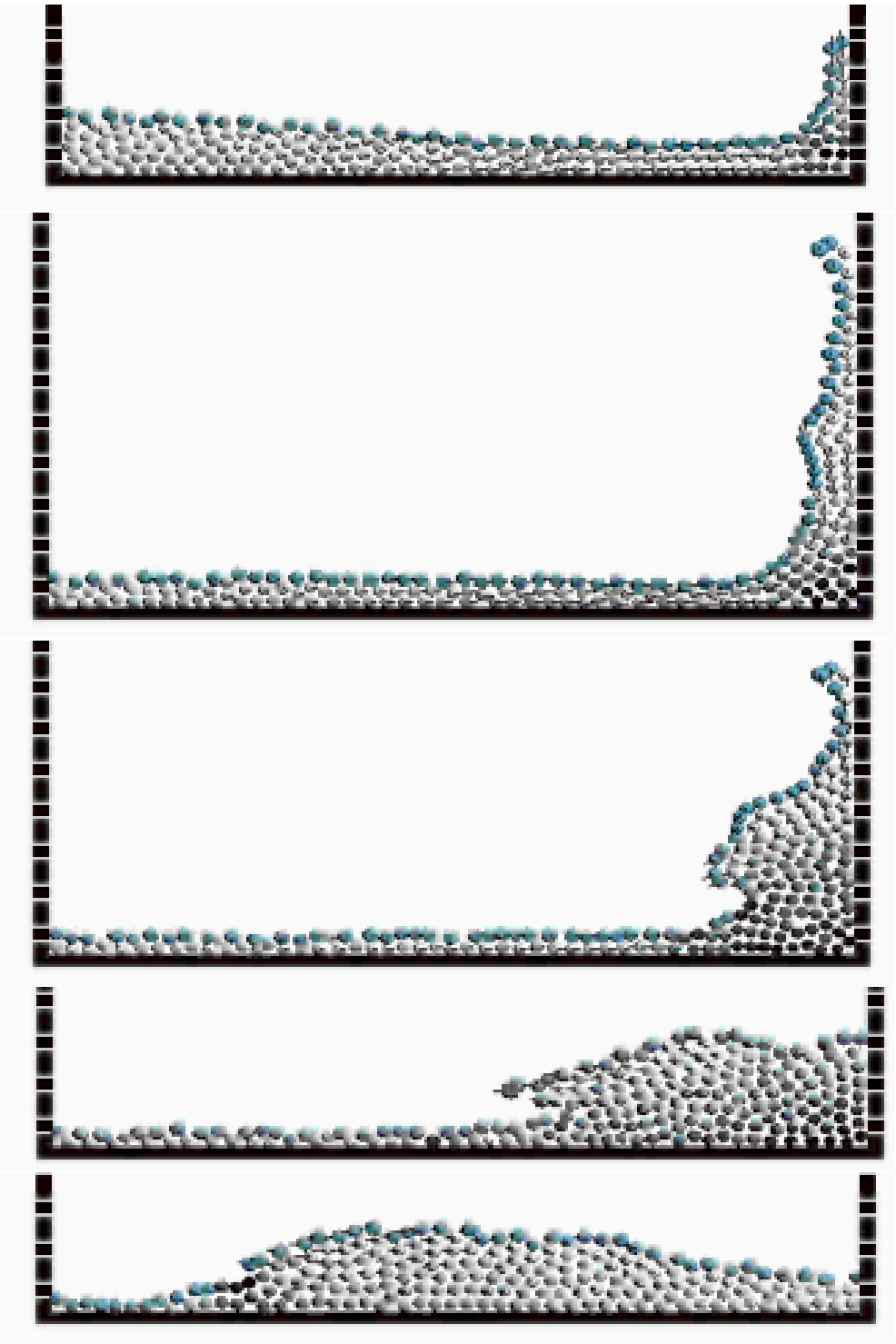


Figure 3.47: Water column hitting a wall using MPS - MCI

B) Lagrange Implicit Fraction Step method and Radial Basis Functions

The water column collapse shows promise for the LIFS method with the inclusion of correct boundary conditions on the fluid. Figure 3.48 shows the pressure distribution (as contour lines) as well as the velocity (as vectors) when $t = 0^+$.

However, the use of Radial Basis Functions proved to be both slow and unstable. Figure 3.49 shows the subsequent decay of the water column. As a particle would approach another particle, the calculations would degenerate because of a singular matrix in Radial Basis Functions.

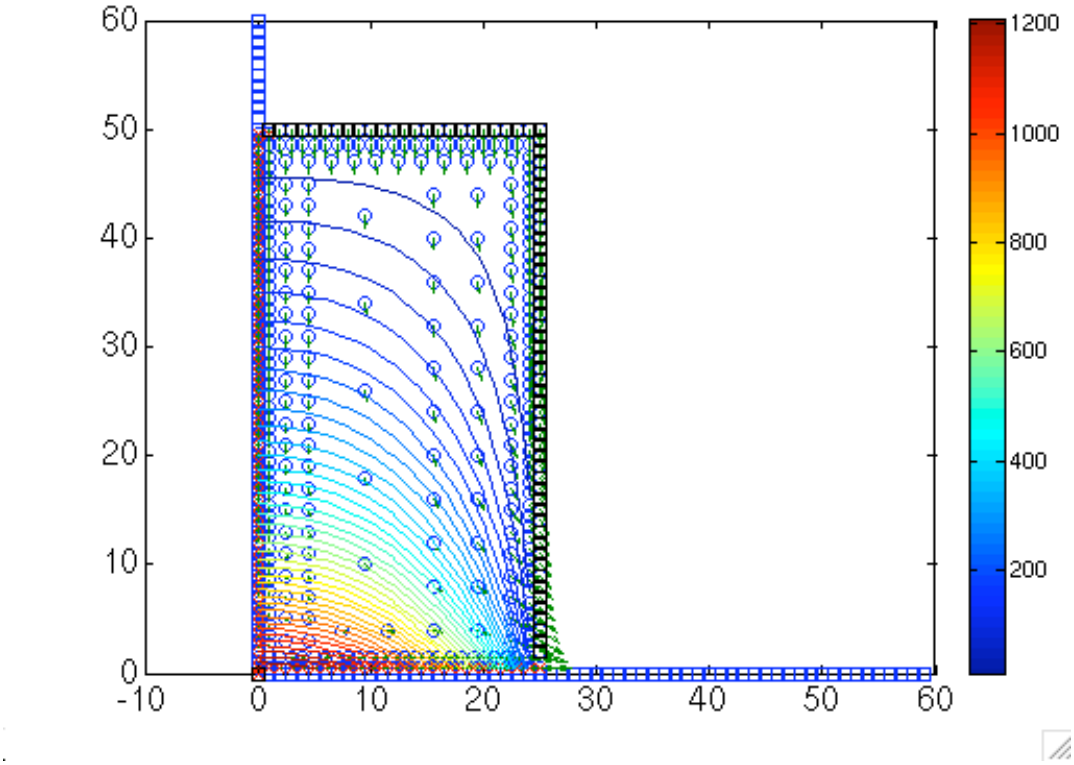


Figure 3.48: Initial pressure distribution in the water column using LIFS - RBF

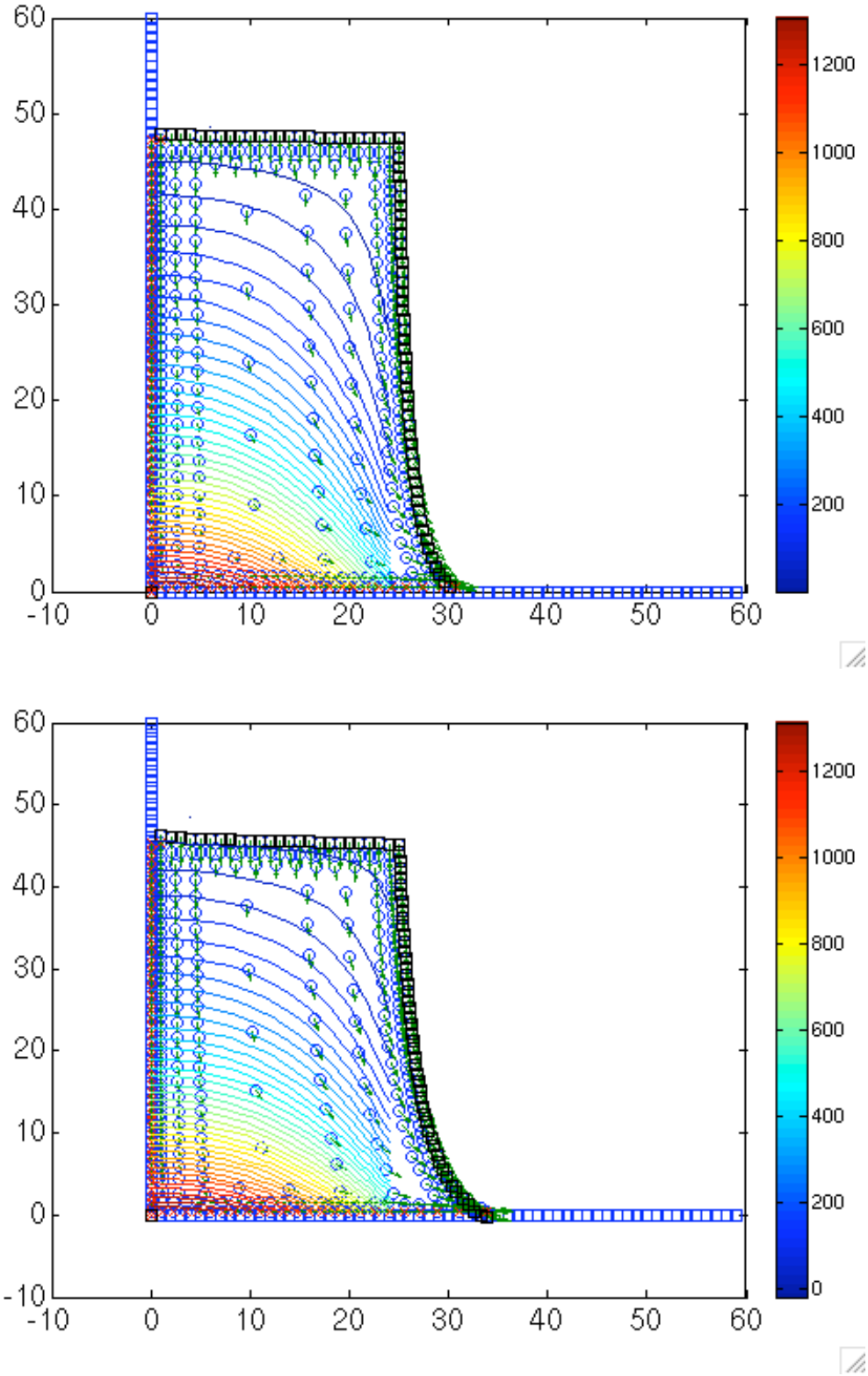


Figure 3.49: Propagation of water column using LIFS - RBF

Finally, the identification of free surface particles was only done at the beginning of the simulation. Because a particle was not re-tagged if a free-surface particle fell into the fluid, a free-surface particle would not obey wall boundaries. Figure 3.50 shows the consequences of not re-identifying boundary particles.

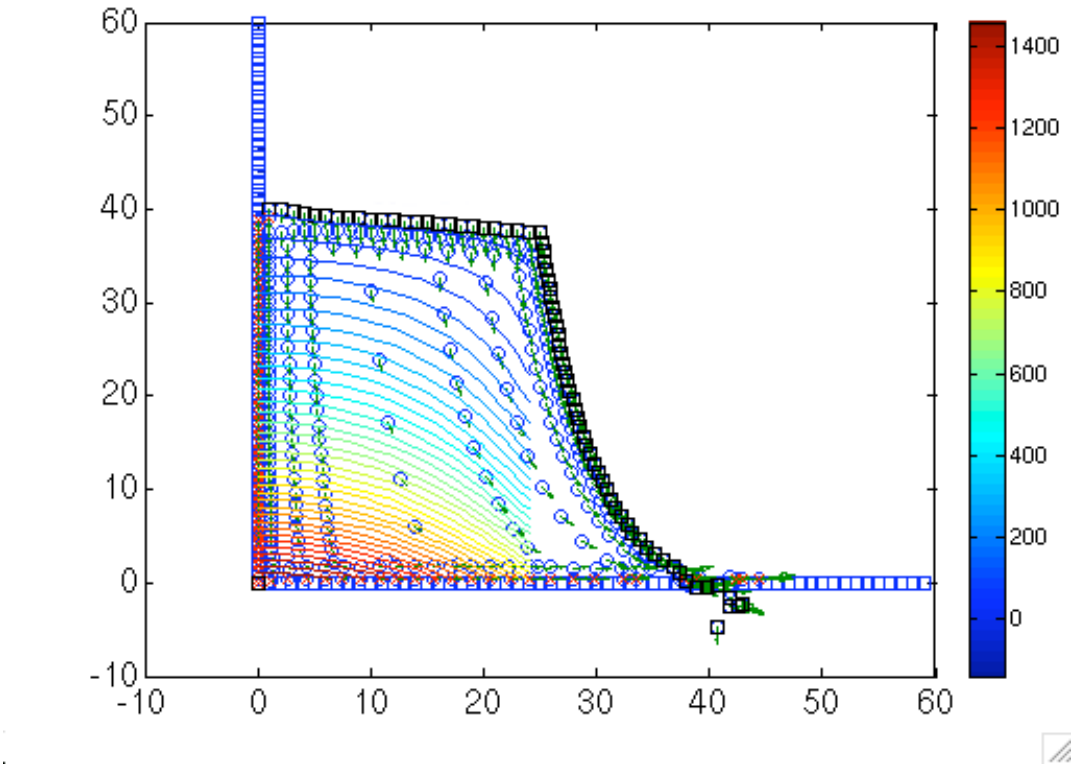


Figure 3.50: Degeneration of LIFS - RBF solution due to boundary condition mistracking

3.2.2 Sloshing Wave

The sloshing wave experiment had an initial condition of particles arranged in a sinusoidal fashion. Because the fluid was inviscid, the oscillations of the fluid should continue in forever. Furthermore, in a given period, the particle arrangement should always regenerate the initial condition.

The specific shape of the initial domain is given by figure 3.51. In the initial position, we can calculate the maximum hydrostatic at $(x, y) = (0, 0)$ as:

$$p = \rho g \left(h_0 + \eta \right) \tag{3.5}$$

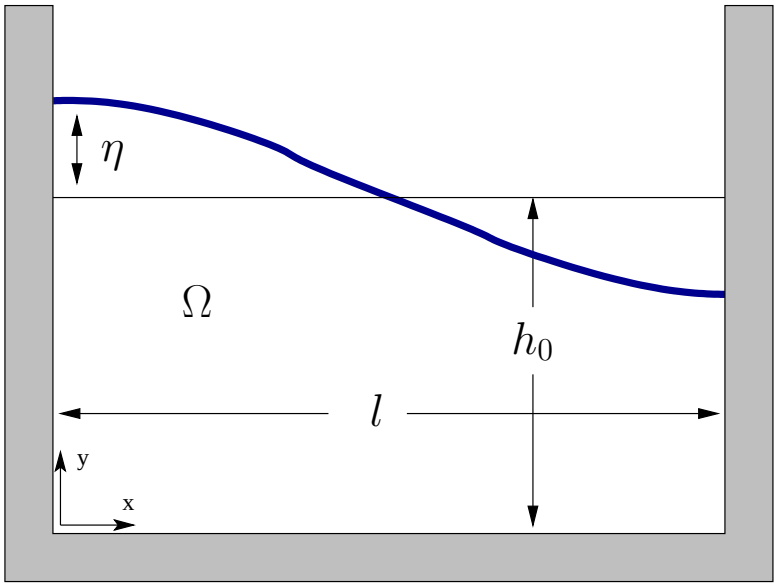


Figure 3.51: Diagram of Sloshing Wave Experiment

A) Moving Particle Semi-Implicit and Monte Carlo Integration

The MPS method showed that there is a large amount of numerical damping in the sine-wave calculations. The simulation also required an unnecessary amount of particles. Because a sparse grid could not be utilized with the Monte Carlo method, particles in lower portion of the domain had to be calculated, but had little consequence on the free surface. These particles remained present so that the Monte Carlo method would not break down. The loss of kinetic energy can be shown in figure 3.52.

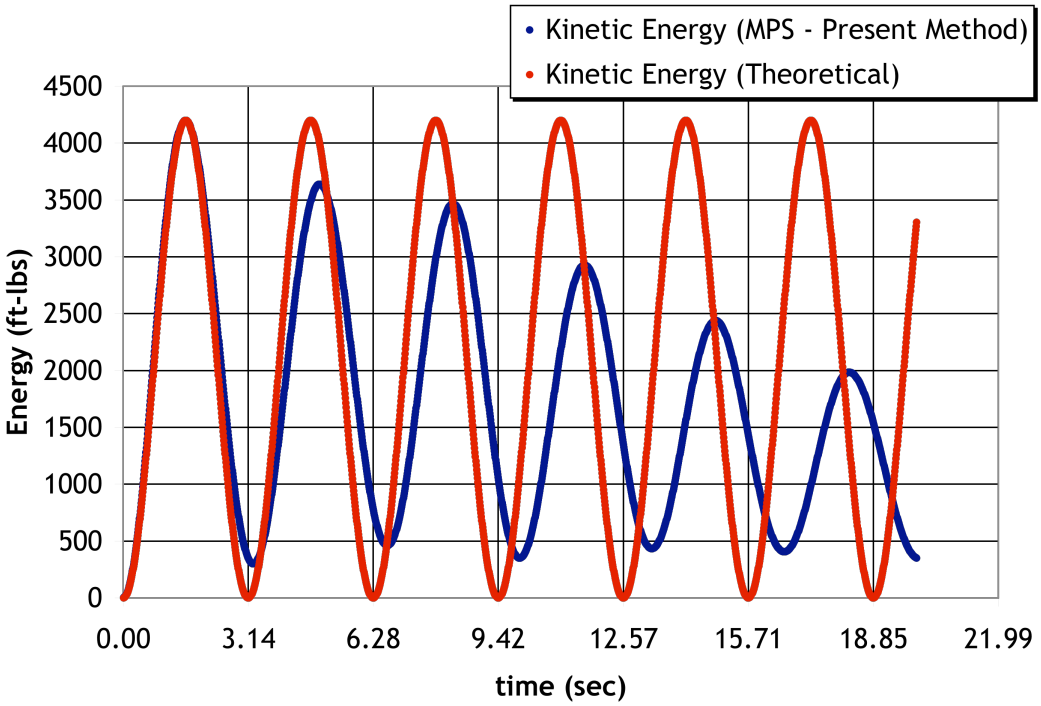


Figure 3.52: Loss of kinetic energy in MPS-MCI in the sloshing wave experiment

Figures 3.53 and 3.54 show the first period, T of the oscillations. In much later oscillations, small vortices were seen in the bottom corners of the domain.

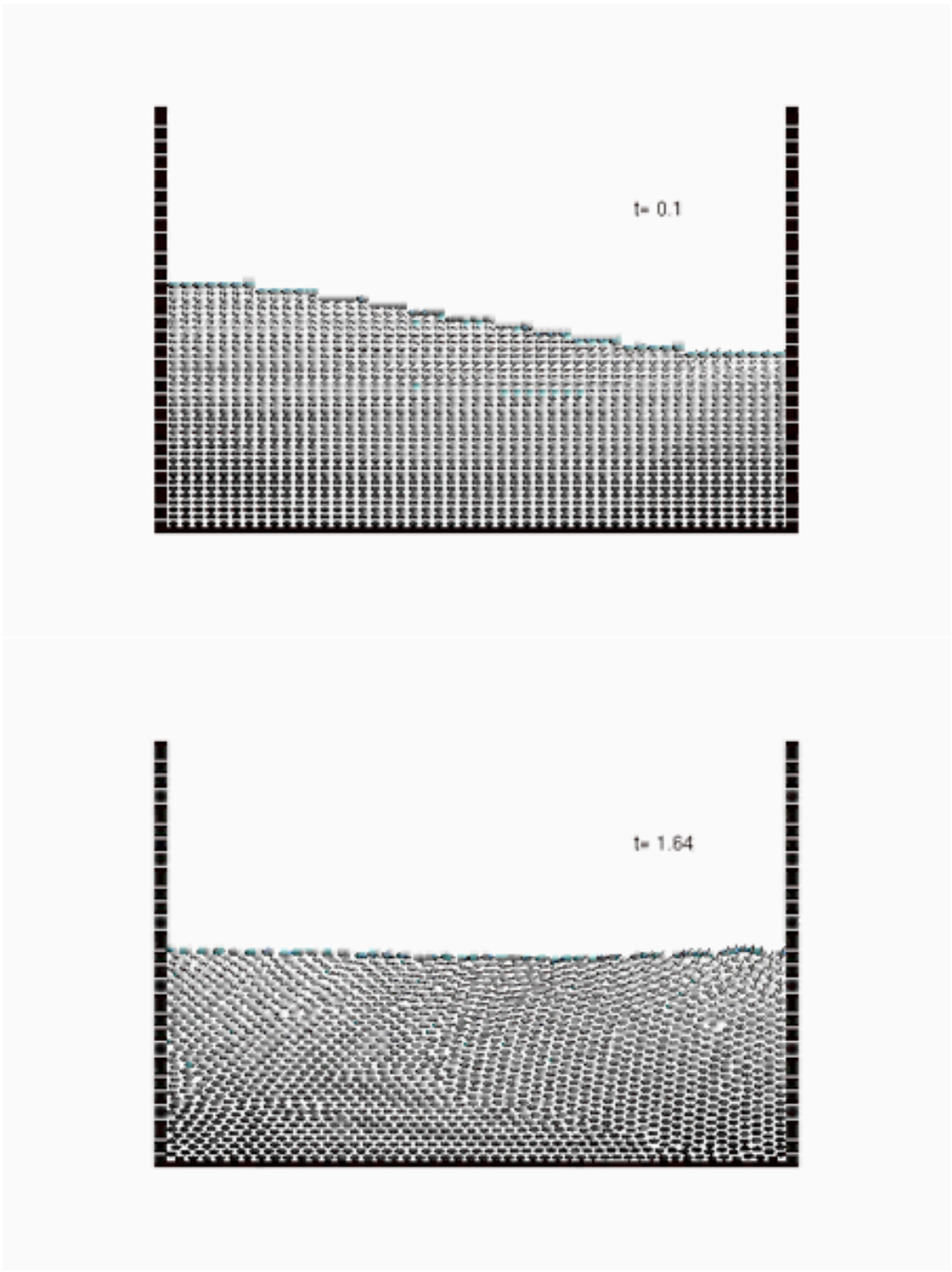


Figure 3.53: MPS-MCI Sloshing solution at $t/T = 0$ (top), $= 1/4$ (bottom)

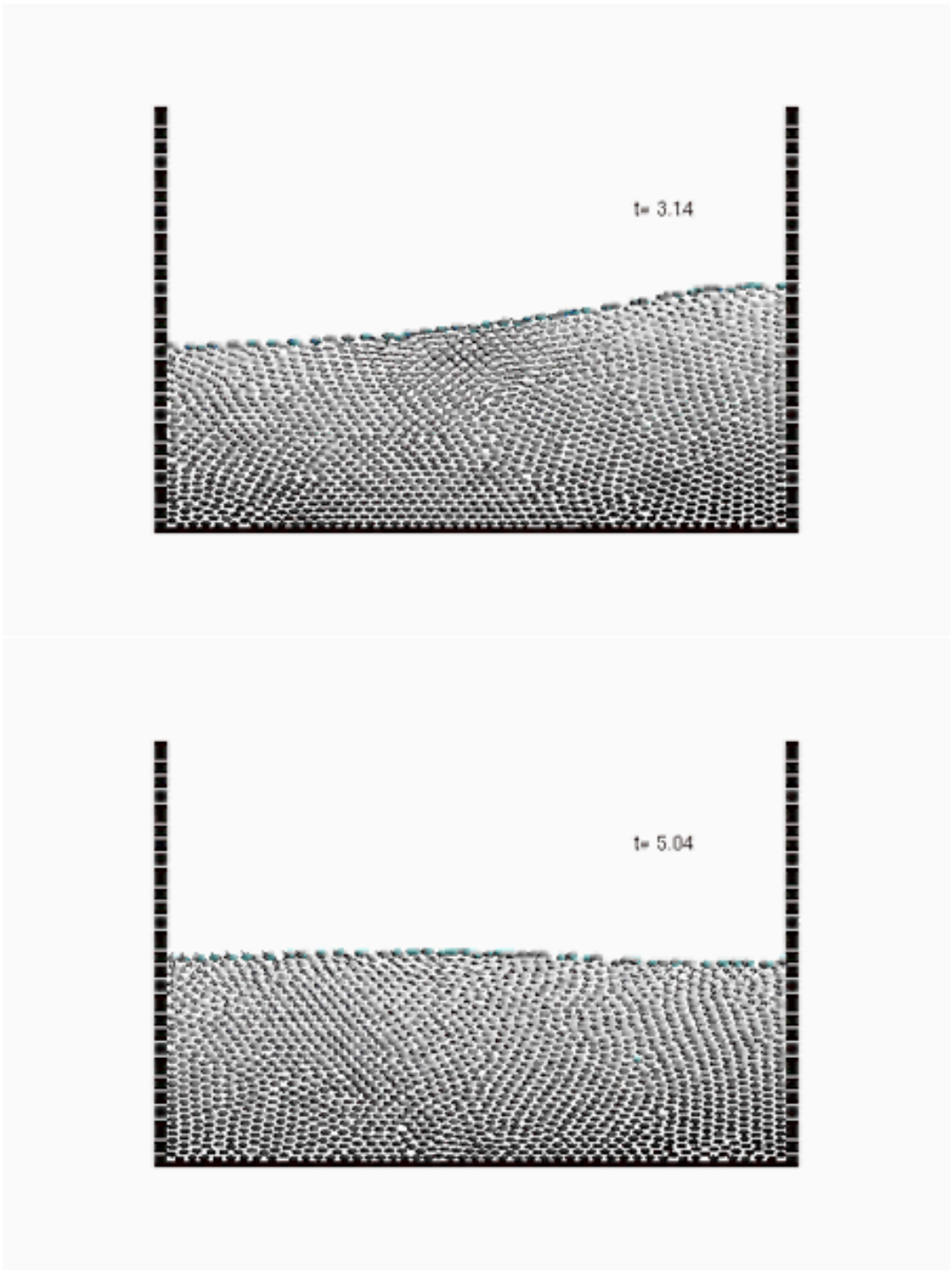


Figure 3.54: MPS-MCI Sloshing solution at $t/T = 1/2$ (top), $= 3/4$ (bottom)

B) Lagrange Implicit Fraction Step method and Radial Basis Functions

The sloshing simulation shows that the algorithm from the LIFS method is viable, however, the use of Radial Basis Functions in large deformation problems is not. The pressure and velocity vectors at time $t = 0^+$ are shown in the upper figure 3.56. The enforcement of flux boundary conditions in the partial differential equation results in a correct calculation of pressure.

Pressure at $(x, y) = (0, 0)$ can be compared between both methods in figure 3.55. Because flux boundary conditions are not enforced, the MPS equations need a number of iterations to correctly calculate the pressure. This occurs when a fluid particle "falls" into a wall particle to such a point that the field equation can correctly reproduce the required boundary force.

The propagation of the sinusoid is seen in figures 3.56.

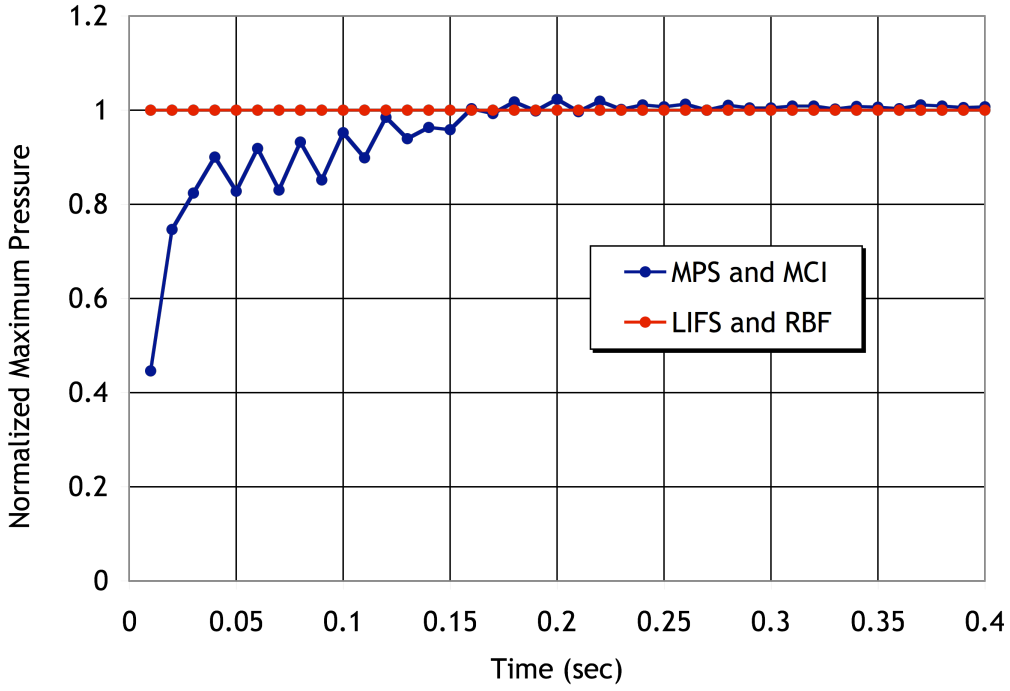


Figure 3.55: Comparison of pressure normalized by hydrostatic pressure eqn(3.5)

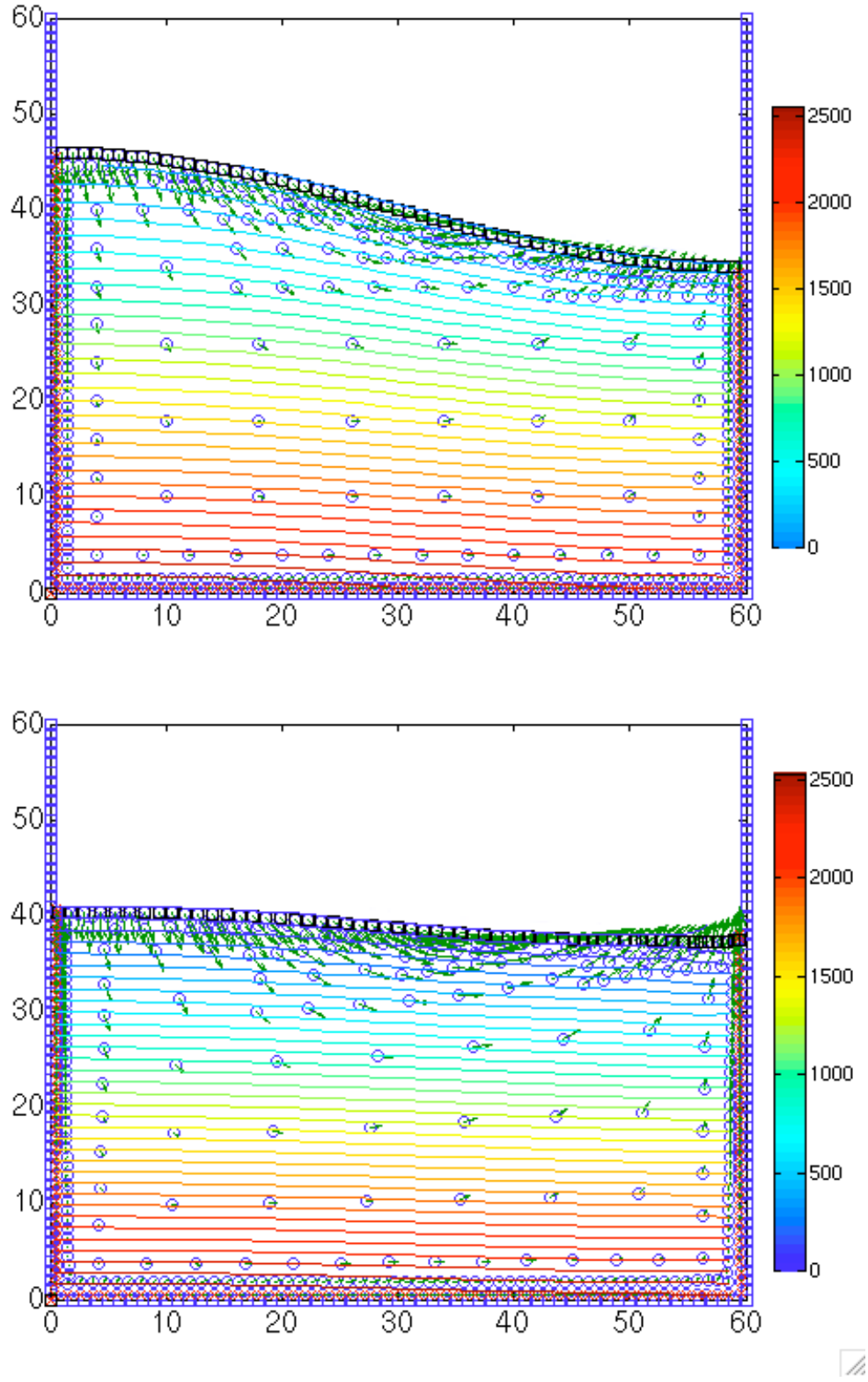


Figure 3.56: Propagation of the sinusoid using Radial Basis Functions

However, after large deformations, the calculations involved with RBFs create the situation that particles in the field equation come too close to each other. The problem with this is that the RBF calculations become unstable, and the simulation breaks down. Figure 3.57 shows the creation of a singularity in pressure near $(x, y) = (60, 40)$.

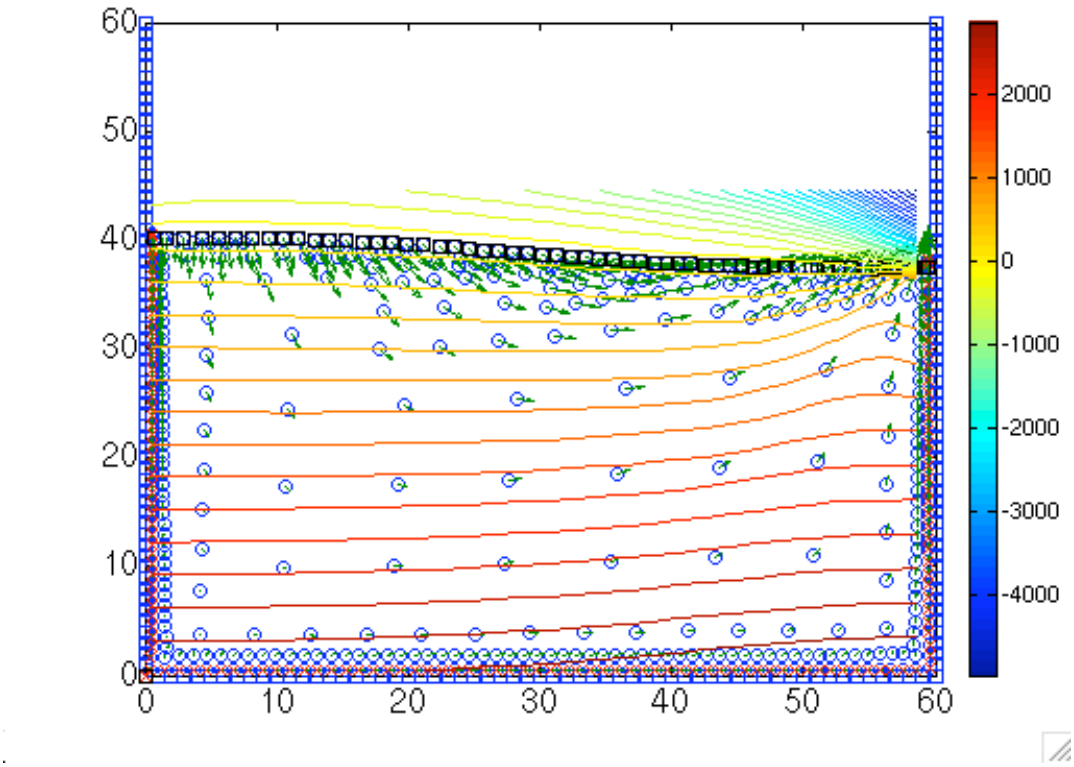


Figure 3.57: Degeneration of solution due to singularity in Radial Basis Functions

3.3 Conclusions

3.3.1 Methods of Computation

The Monte Carlo Integration method posed two sets of problems. First, the calculation of derivatives, especially on the boundaries produced very poor results. Mathematically, derivatives in the MCI method require an infinite domain. If we have a vacancy on any portion of the domain, the calculation of derivatives will not be accurate.

As a consequence, flux boundary conditions from the MCI method cannot be used to solve a system of equations. Therefore, the LIFS method is not usable with Monte Carlo Integration.

Secondly, the MCI method needs a constant spacing of particles to obtain accurate derivatives. This prohibits the use of differently sized particles in a simulation. By not being able to use differently sized particles, the application of the MCI method for any real naval hydrodynamics or full three-dimensional aerodynamics problems is severely limited.

Radial Basis Functions alleviate many of the geometric problems associated with the MCI method. However, the machine costs associated with inverting a full and ill-conditioned matrix limits the amount of particles which could be used in calculations. The cost to invert an RBF matrix can never be less than $O(n^3)$. This cost also prohibits the application of RBFs for any simulation with a large number of particles.

Furthermore, if a grid deformed too much, the matrix could become singular, and the simulation would break down. It is recommended that Radial Basis Functions be used in an Eulerian type simulation where a matrix could be inverted only once per simulation.

Multi-dimensional Lagrange Interpolating Polynomials were created to preserve the nice machine characteristics of the MCI method, while performing more accurate interpolation. The result is a method which was shown to provide accurate interpolation, however, it is relatively untested. The machine expense of performing local-interpolation for each sector is justified by the very low conditioning numbers of the resulting matrices needed to solve a boundary value problem.

3.3.2 Fluid Codes

The MPS equations combined with the MCI method was shown to provide a stable and robust technique. However, the lack of a flux boundary condition for the differential equation meant that the problem was ill-posed. Pressure calculations needed multiple time steps to allow for a fluid particle to "enter" a wall particle. Once the force from the wall-particle sufficiently repelled the fluid particle enough, the pressure calculation would be satisfied. Although the differential equation is ill-posed, the MPS-MCI code still works because the field equation enforced on the wall boundary particles reaches an equilibrium after enough time steps.

Furthermore, in the inviscid calculations, small vortices would occur as a result of smoothing from the MCI technique. Ultimately, this would lead to a decay of energy in the system.

The LIFS equations combined with RBFs showed that the inclusion of flux boundary conditions meant that every time step was capable of correctly calculating the pressure-field. In fact, the LIFS equations differed from MPS only in the initial formulation of the velocity components. By including the velocity components in the calculation of pressure, a well-posed boundary value problem was formed. Furthermore, by separating the influence of viscosity from the other forces, it was possible to form a more efficient inviscid algorithm.

3.3.3 Recommendations and Future Work

Boundary conditions still pose a problem for meshless Lagrangian fluid dynamics. The identification of a particle being either on the free surface or the wall needs to occur for each time step. If not, a free surface particle is capable of "falling-through" a wall, since its pressure is always set to zero. The identification of boundary particles on a sparse, meshless grid is not trivial. One possible solution could be a quadtree "remeshing" of the domain after a certain amount of deformation. While this would be an efficient solution which is also capable of preserving boundaries, this violates a certain trait of the formulation in being meshless.

A second problem is in the free-surface particles. Because all free-surface particles have zero pressure, the pressure gradient perpendicular to the normal of the free surface is always zero. For the breaking dam experiment, this means that the free-surface particles were not capable of supporting each other from the influence of gravity. Instead, they will always fall into each other. We could realize that the pressure for a free surface volume is not actually zero within the volume, but only on the outer surface of that volume. A possible solution could include creating "phantom" free surface particles in an outward normal direction one unit length away. These phantom particles would have zero pressure, allowing the "free-surface" particles to have a pressure able to support one another.

A third problem developed where a particle was both against the wall and on the free surface. In terms of solving the system of equations, a resolution could include using the flux boundary conditions in the system, but while also applying a penalty term in case the pressure was not zero. This problem could also be resolved by the creation of the "phantom particles" for the free surface.

References

Ananthakrishnan, P. and Yeung, R. W. (1994). Nonlinear interaction of a vortex pair with clean and surfactant-covered free surfaces. *Wave Motion*, 19:343–365.

Bonet, J. and Kulasegaram, S. (2002). A simplified approach to enhance the performance of smooth particle hydrodynamics methods. *Applied Mathematics and Computation*, 126:133–155.

Brodkey, R. S. (1995). *The Phenomena of Fluid Motions*. Dover, Mineola, NY.

Chadwick, P. (1999). *Continuum Mechanics, Concise Theory and Problems*. Dover, Mineola, NY.

Cheng, A.-D., Golberg, M., Kansa, E., and Zammito, G. (2003). Exponential Convergence and H-c Multiquadric Collocation Method for Partial Differential Equations. *Numerical Methods for Partial Differential Equations*, 19(5):571–594.

Doring, M., Oger, G., Alessandrini, B., and Ferrant, P. (2002). SPH Simulations of Floating Bodies in Waves. *19th IWWWFB*.

Fedoseyev, A., Friedman, M., and Kansa, E. (2002). Improved Multiquadric Method for Elliptic Partial Differential Equations via PDE Collocation on the Boundary. *Computers and Mathematics with Applications*, 43:439–455.

Gotoh, H. and Sakai, T. (1999). Lagrangian Simulation of Breaking Waves Using Particle Method. *Coastal Engineering Journal*, 41(3 and 4):303–326.

Hildebrand, F. B. (1987). *Introduction to Numerical Analysis*. Dover, Mineola, NY, second edition.

Johnson, B. (1990). On the Integration of CFD and CAD in Ship Design. In G. van Oortmerssen, editor, *CFD and CAD in Ship Design*. Elsevier Science Publishers.

Lo, E. Y. and Shao, S. (2002). Simulation of near-shore solitary wave mechanics by an incompressible SPH method. *Applied Ocean Research*, 24:275–286.

Monaghan, J. J. (1994). Simulating Free Surface Flows with SPH. *Journal of Computational Physics*, 110:399–406.

Newman, J. N. (1977). *Marine Hydrodynamics*. MIT Press, Cambridge, MA.

Shao, S. and Lo, E. Y. (2003). Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface. *Advances in Water Resources*, 26:787–800.

Wang, Z., CPhen, R., Hariharan, N., Przekwas, A., and Grove, D. (1999). A 2N Tree Based Automated Viscous Cartesian Grid Methodology for Feature Capturing. 3300. American Institute of Aeronautics and Astronautics.

Yeung, R. and Ananthakrishnan, P. (1992). Vortical flows with and without a surface-piercing body. *Proceedings 19th Symposium on Naval Hydrodynamics*, pages 219–240.

Yeung, R. W. (1982). Numerical Methods in Free-Surface Flows. *Annual Review of Fluid Mechanics*, 14:395–442.

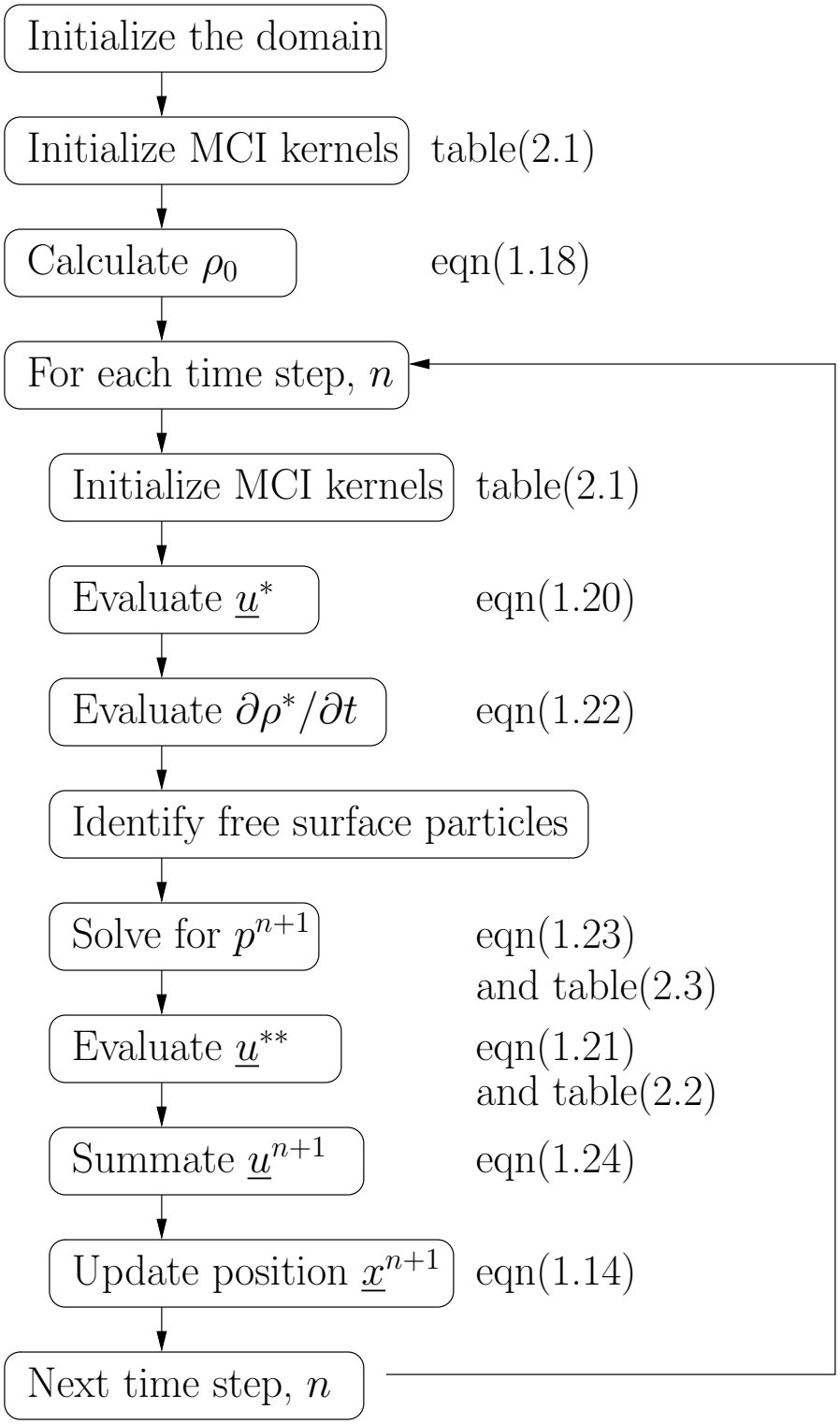
Yoon, H. Y., Koshizuka, S., and Oka, Y. (1999). A Particle-Gridless Hybrid Method for Incompressible Flows. *International Journal for Numerical Methods in Fluids*, 30:407–424.

Appendix A

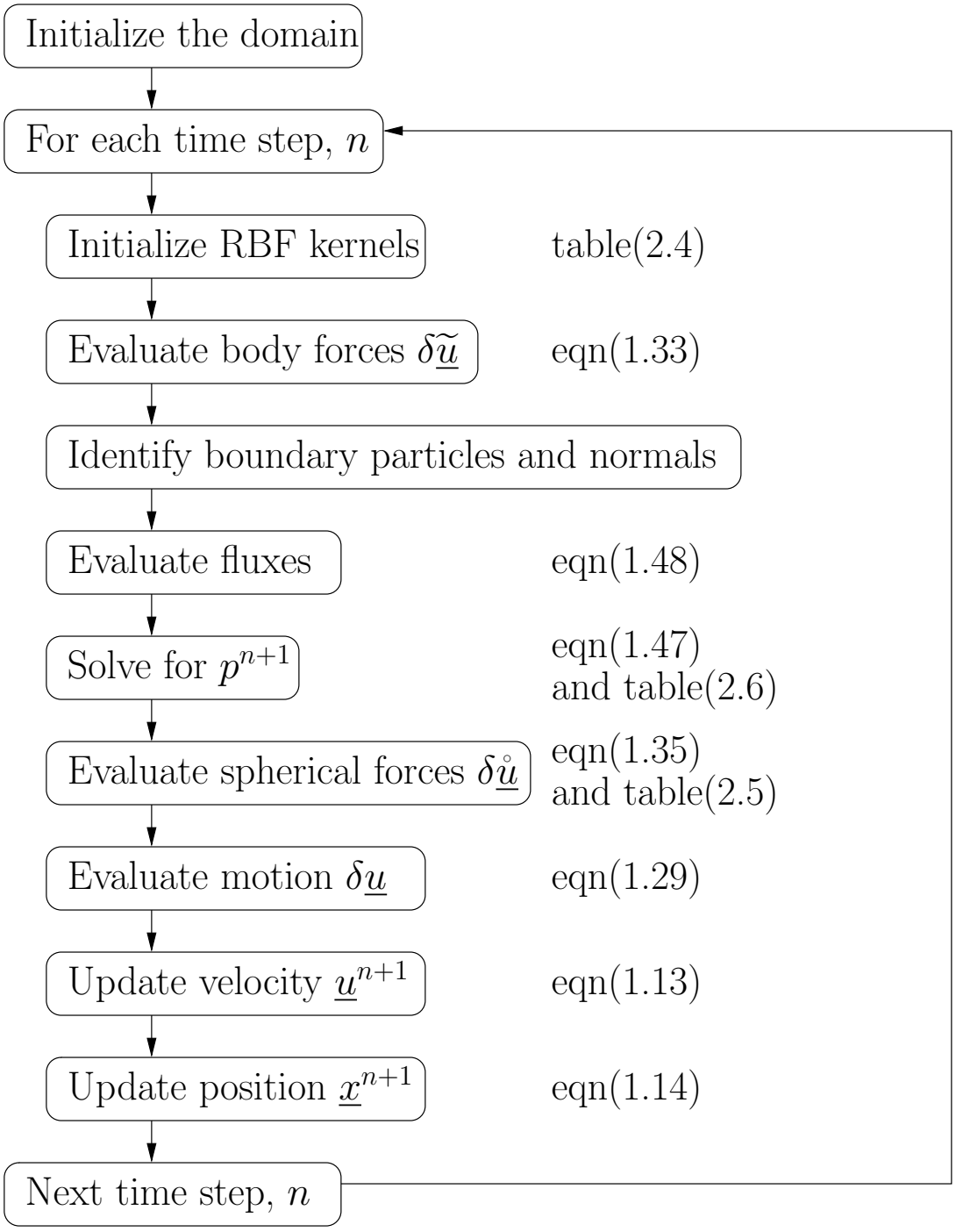
Algorithms

The appendix is divided as follows: Appendix A details the program flow of the MPS-MCI and LIFS-RBF codes. Appendix B details the inputs and outputs to the codes. Appendix C shows the codes contained on the CD.

A.1 MPS-MCI Algorithm



A.2 LIFS-RBF Algorithm



Appendix B

Code Explanation

B.1 MPS-MCI Algorithm

The MPS-MCI code takes an ASCII input file for arguments to the domain and environmental variables. An example file is shown as dam-break.input:

```
-32   .01   4
1     0     2     1     1     10    20
2     0     2     0     0     30    0
2     0     2     0     1     0     30
2     0     2     30    1     30    30
```

On the first line, the first number is the acceleration due to gravity. The second number is the value of δ_t. The third number is the number of "domains."

The subsequent lines are inputs to the "domains." The first number qualifies if the domain is water (1) or a wall (2). The second number is a "modification" to the domain. The modification does nothing when mod=0. The modification changes the domain into a sloshing experiment when mod=1. The third number is the density of each domain.

The following four numbers in the line detail the geometry of each domain. The numbers are inputed as $x0$, $y0$, $x1$, $y1$.

The output from the program are two files. The created file, *out.avi*, is a movie of the simulation. The second created file, *output*, is an ASCII file showing the maximum velocity, maximum pressure, and kinetic energy of the system for every time step.

B.2 LIFS-RBF Algorithm

The inputs to the LIFS-RBF code are written directly into the file *lifsrbf.m*. The variables of δ_t, gravitational acceleration, and density may be modified in line 11 of the file. To perform a sloshing experiment, the lines 17-25 should be uncommented. To perform a dam-breaking experiment, the lines 29-36 should be uncommented. Output is sent to a variable *mov*. This can be turned into an animation by using Matlab's built-in movie to avi function.

Appendix C

File Listing

The CD included with this thesis contains the MPS-MCI code, the LIFS-RBF code, the methods of computation algorithms, and this document. Presently, this CD is also posted at: `www.redcoopers.net`

The folder "MPSMCI" contains the MPS-MCI code:

1. *main.cpp*, and *classes.** are the main computational functions for the MPS-MCI algorithm. The *Numerical Recipes* toolkit is used for mathematical functions and arrays. This toolkit is available for purchase from `www.nr.com`. The GLUT library is also used to display the simulations to the screen.
2. *avi*.**, *opengl.**, and *utility.** are files used to generate movies of the simulations.

The folder "LIFSRBF" contains the LIFS-RBF code:

1. *lifsrbf.m* is the main code.
2. *rbf_kernel.m* and *radius.m* are functions to compute the RBF operators.
3. *sparseinput.m*, *octree.m*, and *uniquer.m* are functions to create a sparse mesh.
4. *printf.** are utilities written in C for Matlab for printing to the screen. *printf.mexmac* must be recompiled for a machine.

The folder "MOC" contains the methods of computation testing algorithms:

1. *grad_2d.m* is the main code.
2. *radius.m* calculates the distances between every particle.
3. *mci_kernel.m* is the code to generate MCI operators.
4. *rbf_kernel.m* is the code to generate RBF operators.
5. *lip_kernel.m* is the main function to generate MLIP operators. It uses the files *liprbf_kernel.m*, *makebins.m*, and *Lagrange*.m* as utilities.
6. *sparseinput.m*, *octree.m*, and *uniquer.m* are functions to create a sparse mesh.