12-19-2008

Development of the Distributed Points Method with Application to Cavitating Flow

David M. Bourg
University of New Orleans

Development of the Distributed Points Method with Application to Cavitating Flow

A Dissertation

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Engineering and Applied Science

by

David M. Bourg

B.S. University of New Orleans, 1992
M.S. University of New Orleans, 1993

December, 2008

ACKNOWLEDGMENTS

I'd like to thank Dr. Robert Latorre for his mentorship and inspiration over my many years at the University of New Orleans. I also thank Dr. William Vorus for his enthusiasm and guidance and for taking over as advisory committee chair upon Dr. Latorre's retirement. Special thanks go to Dr. Juliette Ioup, Dr. Lothar Birk, and Dr. N. Adlai A. DePano for their insight and their participation as advisory committee members post-hurricane Katrina. I'd especially like to thank my wife, Dr. Helena Costales, and my daughter, Natalia, for their support and understanding. Finally, I'd like to thank my parents, Robert and Barbara Bourg, for their encouragement.

TABLE OF CONTENTS

Chapter 1 1
Introduction 1
Background and Motivation 2
Smoothed Particle Hydrodynamics 2
Initial Tests 10
Flow past rectangular obstruction 11
Flow through narrow gap 14
Breaking dam 16
Methods for Dealing with Cavity Flow 19

Chapter 2 23
The Distributed Points Method (DPM) 23
Governing Equations 26
Gradient Computation 30
Least Squares Fit 34
Standard Least Squares Algorithm 34
Bi-quadratic Shape Function Solved Directly 35
Plane Equation Shape Function 41
Two-part Polynomial Shape Functions 43
Weighting Function 45
Numerical Implementation 49
Boundaries 51
Viscosity 52
Nearest Neighbor Search 52
Time Stepping 55

Chapter 3 57
Initial tests of The Distributed Points Method 57
Poiseuille Flow 57
Lid Driven Cavity 63
Multi-fluid Test 67

Chapter 4 74
2D Circular Cylinder Benchmark 74
Baseline Test 75
Parametric Tests 79
Downstream Length Sensitivity 80
Blockage Sensitivity 81
State Equation Sensitivity 81
Weight Function Sensitivity 82
Artificial Mach Number Sensitivity 82
Resolution Sensitivity 82

Reynolds Number Test Results 84

Chapter 5 88
Extension to Multiphase flow 88
Single Phase State Equation 89
Multiphase State Equation 92
The Equations 95
Discussion of Parameters 97

Chapter 6 101
Cavitating flow numerical experiements 101
Circular Cylinder Tests 101
Flat Plate Tests 105
Wedge Model 110
Cavity Length 111
Cavity Pinch 113

Towards the Future 118

References 120

Appendix 128
Selected Source Code 128
TakeStep 128
ComputeAllFields_RK4 130
IntegrateContinuityEquation 135
ComputePressures 135
IntegrateMomentumEquations 136
GetGradients 137
ComputeGradientsBiQuadraticNormalEquations 140
ComputeGradientsPlaneNormalEquations 143

Vita 145

LIST OF FIGURES

Figure 1.1: Flow past obstruction initial wake formation 12
Figure 1.2: Wake development 13
Figure 1.3: Further wake development with eddies 14
Figure 1.4: Piston driven flow through gap 15
Figure 1.5: Formation of "cavities" 15
Figure 1.6: Initial particle configuration and start of motion 17
Figure 1.7: Fluid impacting container wall 17
Figure 1.8: Fluid breaking back on itself 18
Figure 1.9: Initial secondary splash created by fluid-fluid impact 18
Figure 2.1: Illustration of Eulerian Approach 28
Figure 2.2: Illustration of Lagrangian Approach 29
Figure 2.3: Illustration of weighted interpolation 31
Figure 2.4: Two-part shape functions 44
Figure 2.6: Gaussian function 47
Figure 2.7: Linear weight function 48
Figure 2.8: Background grid 54
Figure 3.1: Poiseuille flow between two flat plates 58
Figure 3.2: Lagrangian model 59
Table 3.1: Poiseuille flow parameters 59
Figure 3.3: Lagrangian model, Re = 10.0 60
Figure 3.4: Eulerian model results, Re = 10 61
Figure 3.5: Poiseuille flow results comparison 62
Figure 3.6: Lid driven cavity illustration 63
Figure 3.7: Lid driven cavity set up 64
Figure 3.8: Lagrangian test results 65
Figure 3.9: Lid driven cavity Eulerian model velocity plot 66
Figure 3.10: Lid driven cavity Eulerian model pressure plot 67
Figure 3-11: Mulit-fluid test initial set up 68
Figure 3-12: Multi-fluid test initial set up with tracker particles 68
Table 3.2: Multi-fluid flow parameters 69
Figure 3-12: Density interpolation example 70
Figure 3-13: Multi-fluid results 71
Figure 3-14: Droplet density plots 72
Figure 4-15: Settled fluid 73
Figure 4.1: Circular cylinder problem geometry 75
Figure 4.4: Baseline test drag coefficient time history 78
Figure 4.5: Zoomed-in drag coefficient time history 79
Figure 4.6: Parametric test drag coefficient time histories 80
Figure 4.7: Extended domain downstream 81
Figure 4.8: Test 8 drag coefficient time history 83

Figure 4.9: Test 8 drag coefficient power spectrum........84
Figure 9: Drag coefficient results........87
Figure 5.1: Barotropic state equation graph........90
Figure 5.2: State equation graphs for various Mach numbers........91
Figure 5.3: Multiphase state equation........93
Figure 5.4: Effective sound speed........94
Figure 5.5: Effective Mach number........95
Figure 5.6: Multiphase state equation with $\gamma = 7$........98
Figure 5.7: Effect of cavitation factor........99
Figure 6.1: Cavity formation behind circular cylinder........102
Figure 6.2: Formed cavity behind circular cylinder........102
Figure 6.3: Pinched bubble at cavity trailing edge.........103
Figure 6.4: Pressure field at instant of pinch........103
Figure 6.5: Velocity field at instant of pinch........104
Figure 6.6: Close-up of velocity field........104
Figure 6.7: Flat plate density field, 88s........106
Figure 6.8: Flat plate density field, 97s........107
Figure 6.9: Pressure field at 88s........108
Figure 6.10: Velocity field........108
Figure 6.11: Velocity vector field........109
Figure 6.12: Supercavitating wedge........110
Figure 6.13: Cavity length comparison........112
Figure 6.14: Cavity at 107.5 seconds........113
Figure 6.15: Cavity at 112 seconds........114
Figure 6.16: Cavity at 113.5 seconds........114
Figure 6.17: Cavity pinch sequence........115
Figure 6.18: Pressure field at instant after bubble implosion........116
Figure 6.19: Band of high velocity........117

ABSTRACT

A mesh-less method for solving incompressible, multi-phase flow problems has been developed and is discussed along with the presentation of benchmark results showing good agreement with theoretical and experimental results. Results of a systematic, parametric study of the single phase flow around a 2D circular cylinder at Reynolds numbers up to 1000 are presented and discussed. Simulation results show good agreement with experimental results. Extension of the method to deal with multiphase flow including liquid-to-vapor phase transition along with applications to cavitating flow are discussed. Insight gleaned from numerical experiments of the cavity closure problem are discussed along with recommendations for additional research. Several conclusions regarding the use of the method are made.

Keywords: Distributed Points Method, DPM, Smoothed Particle Hydrodynamics, SPH, David Bourg, computational fluid dynamics, cavitation, Navier-Stokes solver, supercavitation, cavity closure.

Chapter 1

INTRODUCTION

The original aim of this research effort was to develop a *Smoothed Particle Hydrodynamics* (SPH) - based method for simulating unsteady cavitating flow. It was intended that the resulting code would yield new improvements to the SPH method and would provide a new method for simulating unsteady cavitating flow. At the start of this effort in early 2005, an extensive literature survey was conducted that led to the conclusion that simulation of cavitating flow using SPH has never been attempted. An additional aim of this research was to further our understanding of the physics relevant to cavitating flow, specifically supercavitating flow.

The original research plan as detailed in my dissertation prospectus presented to my advisory committee in early 2005 made the statement "As with any new research effort I may encounter unanticipated difficulties or surprises that would require redirecting some effort. I've attempted to foresee what areas will require most attention; however, I've also kept the work plan flexible enough to allow adapting the plan as needs arise." Indeed it wasn't very long into my research effort that one of the most unanticipated events struck – Hurricane Katrina – that resulted in a virtual halt to my research plan.

I can remember sitting in a small office in the Veterans Affairs Hospital in downtown New Orleans coding away on what eventually became the result of this research effort when the hurricane struck. In spite of the weather I was making steady progress until the power went out, my laptop batteries died, and the city flooded including the building I was in. That was the end of my research, for a time. When the dust settled, so-to-speak, and I began to pick up the pieces I realized the break in my effort resulted in a somewhat refreshing pause (if you can image recovering from Katrina as being a refreshing pause for a moment) that allowed me to see the problems I was trying to tackle in a renewed light. That is to say, without being buried in all the coding details, I was able to see the big picture clearer than before and realized the path I was headed down would result in a sort of Frankenstein code. What I mean by this is that I was taking the SPH method and attempting to patch over or work around its inherent limitations, by adding further modeling approximations each with their own errors and limitations, which themselves could conceivably require additional workarounds to deal with approximation errors. Instead of heading down that path I decided with my advisor at the time, Dr. Robert Latorre who has since retired from the University, to take a fresh look at the root of SPHs difficulty, namely, the very foundation of the method, it's interpolation scheme.

This refocused effort has yielded an SPH inspired method I call the *Distributed Particle Method* (DPM), which, like SPH, is a meshless method for solving the Navier-Stokes equations. The DPM retains the key benefits of SPH. Further, as originally aimed for, I extended the DPM method to deal with cavitating flow.

This document provides an overview of my research effort summarizing the SPH method as a basis for the DPM method, development and benchmarking of the DPM method for single phase flow, and extension of the DPM method to cavitating flow. The remainder of this chapter gives an overview of the SPH method along with motivation for development of a meshless method capable of handling cavitating flow. Chapter 2 covers details of the DPM as inspired by the SPH method. The next chapter, Chapter 3, describes and presents results for several initial numerical experiments using the DPM. Chapter 4 presents discussion and results for further benchmarking the DPM for 2D flow around a circular cylinder. Chapter 5 explains how the single phase DPM method was extended to treat multi-phased flow in order to simulate the liquid-to-vapor phase transition in cavitating flow. Finally, Chapter 6 presents results from several numerical experiments conducted using the multi-phase DPM.

Background and Motivation

Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics was originally developed by J. J. Monaghan in 1977 to solve astrophysical problems. Originally, the method was developed to solve 3D compressible flow problems and was later adapted for 3D and 2D incompressible flow problems. Indeed, the method has shown its great flexibility over the last few decades in that it has been successfully adapted to solve a wide variety of problems from astrophysics, to ocean wave dynamics, to internal flow in machinery, to blood flow through arteries, to interactive entertainment visualization, to underwater explosion, and solid impact problems. This adaptability is truly a testament to the method's generalized nature and versatility.

SPH is a meshless, Lagrangian particle method that is capable of handling free surface problems just as easy, if not easier, than internal flow problems. SPH was originally developed for boundless problems and treatments of rigid boundary conditions were added to the method over many years of development and adaptation to a variety of problems. That said, rigid boundaries still remain one of the more challenging aspects of implementing SPH owing to the truncation error, among other phenomenon, that results when field variables are interpolated over disparate points asymmetrically located adjacent to a rigid boundary.

Because SPH is a particle-based method instead of a mesh-based method, it is easier to code, especially for free surface problems where mesh entanglement is a serious issue with mesh-based methods. In SPH, once the governing equations are developed for the fluid(s) involved, the solution comes down to integrating the equations of motion for a number of particles representing the fluid(s). Implementation of SPH becomes more difficult when CPU usage optimization is required. Naive implementations of SPH result in tremendous computation requirements as the method is an N-Body method where, theoretically, every particle depends upon every other particle in the simulation. Thus, computationally speaking the method scales as $O(N^2)$ as the number, N, of particles is increased. There are several approaches taken in practice to reduce the complexity to something approaching O(2N). (These approaches involve using kernels with compact support and implementation of optimized nearest neighbor search algorithms.)

The method of SPH is explained in detail in references [11], [13], and [14], among others, so I won't repeat all the details here. However, I'll give a brief overview of the fundamental principles of the method.

Monaghan [11] states "at the heart of SPH is an interpolation method which allows any function to be expressed in terms of its values at a set of disordered points – the particles." This is a key feature of SPH that makes it so flexible. In grid based methods, quantities are interpolated across grid points, or nodes, whereas in SPH the particles themselves represent nodes across which values can be interpolated.

In SPH, the integral interpolant of a function *A(r)* is defined by the following:

$$A(\bar{r}) = \int A(r')W(\bar{r} - \bar{r}', h)d\bar{r}'$$

or,

$$A(\bar{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\bar{r} - \bar{r}_j, h)$$

where *m* is the mass of particle *j*, *A* is the value of any quantity of interest, r_j is the position vector of particle *j*, ϱ_j is the density of particle *j*, and *W* is the interpolation kernel which is a function of *r* and *h*. Here *h* is the so-called *smoothing length*, which is ½ the radius which defines the circle encompassing nearest neighbors for a given particle. The smoothing length defines the support domain for the interpolation kernel of compact support. Another way of looking at the smoothing length is as the equivalent to the *grid-scale* of mesh-based methods.

The kernel interpolation function is typically Gaussian, or a close approximation, where only particles located within the smoothing circle contribute to the value of a quantity for a given particle. All particles falling outside this circle don't contribute. This is key to optimizing the SPH method in terms of CPU usage as will be discussed later. (The Gaussian interpolation kernel does not have compact support; therefore, other kernels have been designed for specific applications that approximate the Gaussian but possess compact support.)

What's clever about this method is that the kernel is an explicit function that can be differentiated analytically. This means that gradients of quantities of interest can be expressed in terms of the gradient of the kernel as follows:

$$\nabla A(\bar{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\bar{r} - \bar{r}_j, h)$$

Thus, you don't have to take finite differences as you do in mesh based methods in order to calculate quantity gradients. The trick, however, is designing kernels that approximate the Gaussian kernel well enough but possess compact support and possess continuous first and second (or higher in some cases) derivatives.

Many different forms of the SPH interpolation kernel are in use these days and [24] describes specific requirements that must be fulfilled when designing an interpolation kernel.

For illustration purposes, I've included the cubic spline formulas here since this kernel seems to be the most widely used:

$$W(r,h)=\left\{\begin{array}{l}\frac{\sigma}{h^2}\left(1-\frac{3}{2}s^2+\frac{3}{4}s^3\right),0\le s<1\\ \frac{\sigma}{h^2}\left(\frac{1}{4}(2-s)^3\right),1\le s<2\\ 0,s\ge 2\\ \end{array}\right\}$$

where,

$$s=\frac{|\bar{r}|}{h}$$

$$\sigma=\frac{10}{7\pi}$$

As you can see, this kernel is effective for a distance of *2h* around the particle under consideration. Any particles falling outside this radius will not contribute to the interpolation.

SPH allows various equations of state for the fluid(s) under consideration. For incompressible flow problems the following equations of state have been used:

$$P=c^2\rho$$

$$P = \frac{c^2 \rho_o}{\gamma}\left[\left(\frac{\rho}{\rho_o}\right)^{\gamma} - 1\right]$$

Here, *P* is the dynamic pressure of the particle under consideration, *c* is the speed of sound, ρ_o is the reference (typically the initial) density, *γ* is the specific heat ratio, and *ρ* is the current density of the smoothed particle.

In SPH, pressure is an explicit function of density as you can see from the above equations. Recall that originally SPH was developed for compressible flow problems, thus to apply the method to incompressible flow problems, you have to use an artificial speed of sound, *c*, such that density variations are kept to about 1% to 3%; this gives a Mach number based on the maximum expected fluid velocity of about 0.1. Monaghan has shown, through analysis of the Navier-Stokes equations that the proper choice of *c* can be obtained from the following:

$$c^2 = \max\left[\left(\frac{V_o^{\ 2}}{\delta}\right), \left(\frac{\upsilon V_o}{L_o \delta}\right), \left(\frac{F L_o}{\delta}\right)\right]$$

where

$$\delta = \frac{\Delta\rho}{\rho_o}$$

Here, V_o is the velocity scale, L_o is the length scale, *F* is the body force per unit mass acting on the fluid, and *ν* is the kinematic viscosity.

Recently, truly incompressible versions of the SPH method have been introduced [25]. These versions eliminate the state equation and instead solve a Poisson equation to obtain pressure. Further, these versions require intermediate steps to evolve particle motions and then correct resulting velocities to yield a divergence free velocity field.

In SPH, the density of each particle can be calculated using the following formula:

$$\rho_i = \sum_{j=1}^{N} m_j W_{ij}$$

Here, the density of particle *i* is found by interpolating over all other particles *j*. While this formula is simple in form and conserves mass, it does require multiple iterations over all particles before each integration time step in order to determine the density of each particle. A more computationally efficient approach is to use the SPH form of the continuity equation and evolve particle density during the simulation:

$$\frac{d\rho_i}{dt} = \sum_{j=1}^{N} m_j (\bar{v}_i - \bar{v}_j) W_{ij}$$

Here, *v* is the particle velocity. This equation allows you to update the density (starting from an initial density) of each particle at each time step, thus it does not require extra iterations through the entire list of particles as does the previous equation for density. That said, this method of calculating density does not conserve mass exactly.

The SPH form of the momentum equation, without viscosity, is as follows:

$$\frac{dv_i}{dt} = -\sum_{j=1}^{N} m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2} \right) \nabla_i W_{ij}$$

This is essentially the SPH version of Euler's equation for a fluid particle. Fluid particles are driven by pressure gradients as you can see from the above equation. This equation is symmetric in that the pressure gradients result in equal and opposite central forces acting on particle pairs. Thus, linear and angular momentum are conserved.

The addition of viscosity adds more terms to the above momentum equation. Essentially viscous forces are converted to equivalent pressure forces and included in the inner sum of pressures in the momentum equation.

Once the change in particle velocities have been determined using the momentum equation, their positions can be updated using the following equation:

$$\frac{d\bar{r}_i}{dt} = \bar{v}_i + \varepsilon \sum_{j=1}^{N} m_j \frac{\bar{v}_i}{\bar{\rho}_{ij}} W_{ij}$$

Here, *r* is the position vector for each particle, *v* is the velocity, $\bar{\rho}_{ij} = (\rho_i + \rho_j)/2$, and *ε* is a constant that ranges from 0 to 1. The term on the right of the plus sign represents the XSPH correction for velocity [13]. It isn't required, but does tend to make particle motion a bit more orderly, which is desirable for high speed incompressible flow problems that involve free surfaces.

In Monaghan's original development of SPH he uses an artificial viscosity to introduce shear and bulk viscosity into the momentum equation. As I stated earlier, in his method, viscous forces are converted to equivalent pressure forces that are included in the SPH momentum equation as follows:

$$\frac{dv_i}{dt} = -\sum_{j=1}^{N} m_j \left(\frac{P_j}{\rho_j^{\ 2}} + \frac{P_i}{\rho_i^{\ 2}} + \Pi_{ij} \right) \nabla_i W_{ij}$$

where,

$$\Pi ij = \begin{Bmatrix} \dfrac{-\alpha \bar{c}_{ij} \mu_{ij} + \beta \mu^2{}_{ij}}{\bar{\rho}_{ij}} ; \bar{v}_{ij} \bullet \bar{r}_{ij} < 0 \\ 0; \bar{v}_{ij} \bullet \bar{r}_{ij} > 0 \end{Bmatrix}$$

$$c_{ij} = (c_i + c_j)/2$$

$$\bar{\rho}_{ij} = (\rho_i + \rho_j)/2$$

$$\mu_{ij} = \frac{h\bar{v}_{ij} \bullet \bar{r}_{ij}}{r^2{}_{ij} + 0.01h^2}$$

For most incompressible flow problems, β is set to 0 while a is set to 0.01. This makes the viscosity purely shear as the β term is for bulk viscosity. Artificial viscosity was introduced into SPH to solve problems involving shocks. Further, some practitioners like to introduce artificial viscosity as a means of providing some damping to improve numerical stability. The empirical coefficients appearing in the artificial viscosity formula are tuned to suit the problem under investigation.

SPH is capable of incorporating physical viscosity as well [24][26][27][28]. There are essentially two ways to handle physical viscosity in SPH. One approach is the straightforward application of the second derivative of the interpolation kernel to formulate an SPH version of viscous terms that appear in the Navier-Stokes equations. This approach has been used with some success. Care, however, must be taken as results tend to be sensitive to particle disorder and number density; thus, "remeshing" schemes are typically used to redistribute particles throughout the simulation to keep them in order.

Another approach involves applying a Taylor series expansion along with the first derivative of the interpolation kernel to approximate the second derivative. This approach works too; however, it requires nested summations over particles, which increase computational costs. Both of these approaches add coding complexity and computational overhead; thus, most practitioners seem to avoid their use and opt for artificial/empirical models for viscosity which are tuned for the given problem under investigation. (The DPM avoids these difficulties by directly approximating the viscous terms in the Navier-Stokes equations with 2nd order interpolation functions that can be differentiated analytically.)

Boundary conditions are typically handled by representing rigid boundaries with smoothed particles that are fixed in position [8][13][14][24]. Further, these boundary particles are modeled to exert a strong repulsive force on free flowing particles that come into contact with the boundary. This approach satisfies the no flow through boundary condition. To model the no slip boundary condition, the boundary particles include one or more layers of so-called "ghost" particles, which get included in the viscous force summations.

Modeling boundaries in this way makes it relatively easy to simulate arbitrarily shaped boundaries. One need only distribute boundary particles over rigid surfaces to approximate their shape. However, this approach is not without it's problems. First, using a repulsive force model (usually a Lenard-Jones model) can sometimes lead to very strong forces imposed on particles that get too close to a rigid boundary. This can lead to severe numerical instabilities and unphysical results. The strength of the repulsion force must be carefully tuned to mitigate these effects. Second, distributed particles along a boundary can lead to an essentially "bumpy" boundary rather than a smooth one, which tends to result in unrealistically high viscous drag. Further, unless special care is taken [29] viscous forces imposed on fluid particles by boundary particles are not purely shear and include a normal component that tends to unrealistically push fluid particles away from boundaries.

Initial Tests

As part of my research effort, I prepared an SPH implementation using standard SPH techniques for solving incompressible flow problems. This implementation uses the equations discussed earlier along with the standard artificial viscosity models. It also uses distributed boundary particles with a Lenard-Jones type repulsion force. (As discussed earlier, this approach does have some pitfalls.) Further, for this implementation I've designed a linked list, nearest neighbor, tracking algorithm that calculates, during each time step, all the particles that are within 2H of each particle. This nearest neighbor information is stored in a series of linked lists that are then traversed during integration. While the nearest neighbor search adds an additional loop through the particle list, the savings in terms of CPU time are dramatic. With this algorithm, the integration goes from $O(N^2)$ to something approaching O(2N) in terms of speed as a function of particle number. This is a crucial element of this implementation as simulations of reasonable resolution can be performed on standard, scalar desktop computers.

The aim of this initial implementation was primarily as a learning exercise to become intimately familiar with the SPH method. During the course of developing this initial program I used the breaking dam problem as a test case. Reviewing the SPH literature gives the impression that the breaking dam problem is almost a right of passage for anyone making their foray into SPH. I suppose one of the reasons for this is that the problem looks rather impressive to the un-initiated. From a more technical perspective, the breaking dam offers the ability to test rigid as well as no slip boundaries in addition to free surface flow and splashing, thus it's a good problem to check to make sure the code is stable and working properly. Further, the breaking dam problem is a common enough benchmark for SPH that there are several sources of comparison simulations. There's even a source [23] that shows physical test results of the breaking dam problem, which has proved to be a valuable basis of comparison.

As part of this preliminary development effort, I modeled other flow problems, namely internal flow problems. These simulations consist of piston driven flow through a narrow passage. In one case the passage is obstructed with a rectangular block that forces fluid to flow through two narrow gaps on either side. In the other case the passage is blocked by a vertical plate that forces fluid to flow through a single narrow gap. Both of these problems are more relevant to the subject of the proposed research and are discussed more in the following paragraphs.

In both cases, preliminary results agree qualitatively with available experimental results and clear separation points are observed. Further, flow structures in the wake region of the rectangular obstruction problem are consistent with experimental observations. Simulations of flow through the narrow gap showed interesting results that warranted further study and which was, in part, motivation for this research effort.

Flow past rectangular obstruction

Figures 1.1, 1.2, and 1.3, show some results of piston driven flow of an incompressible fluid through a narrow passage and around a block-shaped obstruction. These figures show particle traces with trails representing particle path lines over several time steps. The color scale represents pressure with blue indicating low pressure and red indicating high pressure. The moving piston was modeled as rigid body with prescribed motion from left to right. Interaction between the piston boundary and particles was modeled using distributed boundary particles.

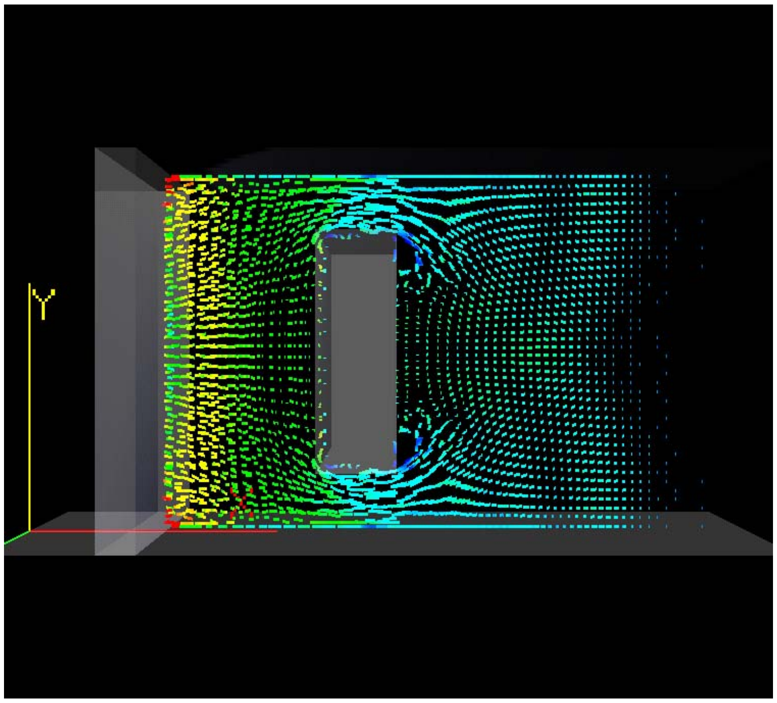


Figure 1.1: Flow past obstruction initial wake formation

Figure 1.1 shows the piston driven flow at a point shortly after the piston was impulsively started (from left to right). Flow separation at the corners of the obstruction are clearly evident. The figure also shows a leading pressure wave that has diffracted around the edges of the obstruction.

Figure 1.2 shows the same simulation a few time steps after that instant shown in Figure 1.1. Here you can observe the formation of eddies in the wake of the obstruction.

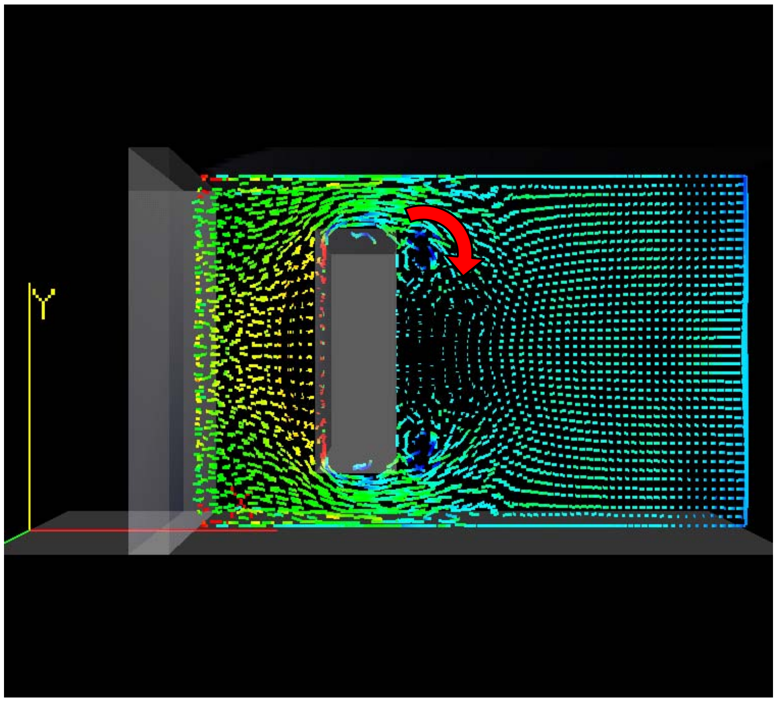


Figure 1.2: Wake development

A few time steps later, the wake is better developed as illustrated in Figure 1.3. The arrows in Figure 1.2 and 1.3 indicate the direction of flow. An important feature captured rather well by SPH is the pressure distribution whereby the pressure is low as expected at the centers of the eddies in the wake. Further, as expected, the pressure is high on the leading face of the obstruction.

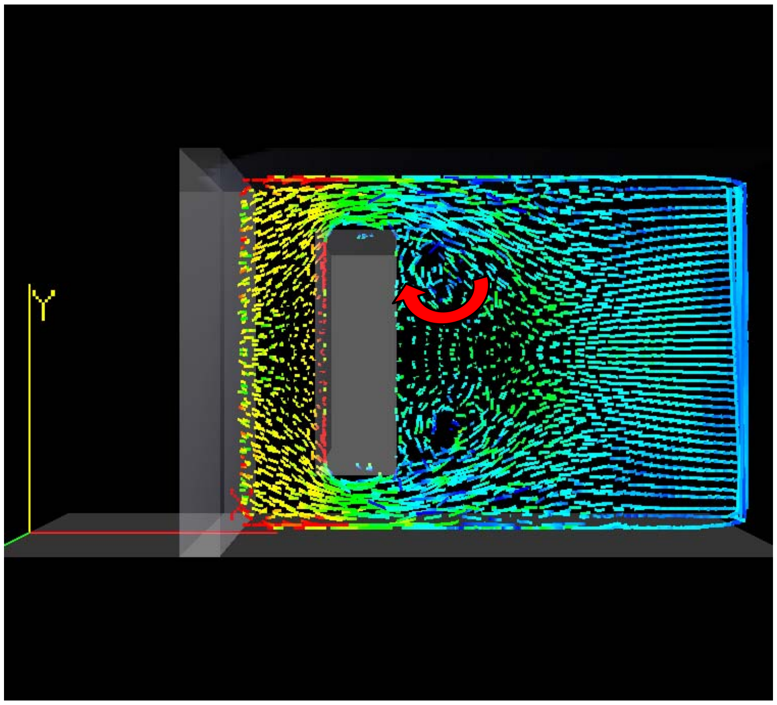


Figure 1.3: Further wake development with eddies

In all of these figures, separation at the sharp leading edge corners of the block can be observed. While it isn't clear in these images, there exists a small region of re-circulating flow behind the separation points. Particles flow around these re-circulating regions and accelerate through the narrow gaps between the block and the channel walls.

Flow through narrow gap

Figures 1.4 and 1.5 show results from a similar problem. In this case, the obstruction blocks most of the channel except for a small gap at the upper end. Here the piston was driven much faster than in the previous case and a finer particle distribution was used (along with smaller time steps). The color scale represents particle speed with red being faster moving particles and blue being slower moving particles.

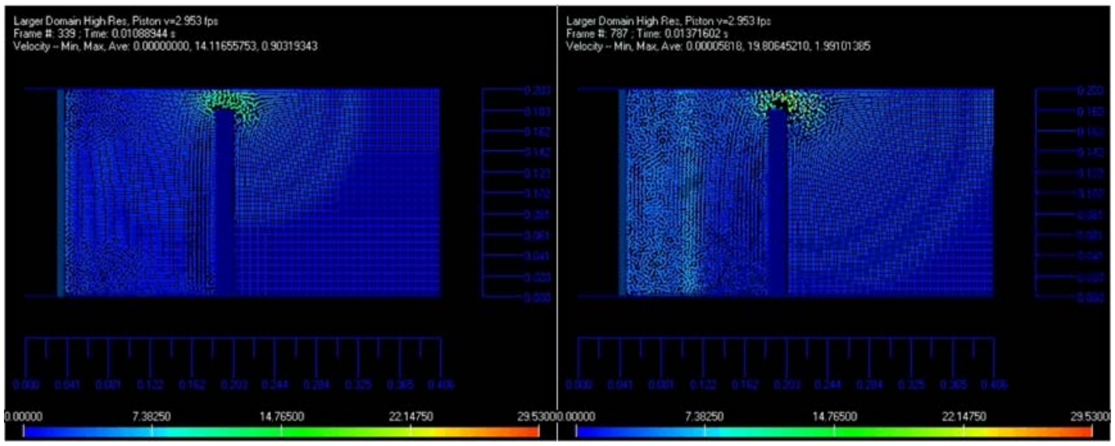


Figure 1.4: Piston driven flow through gap

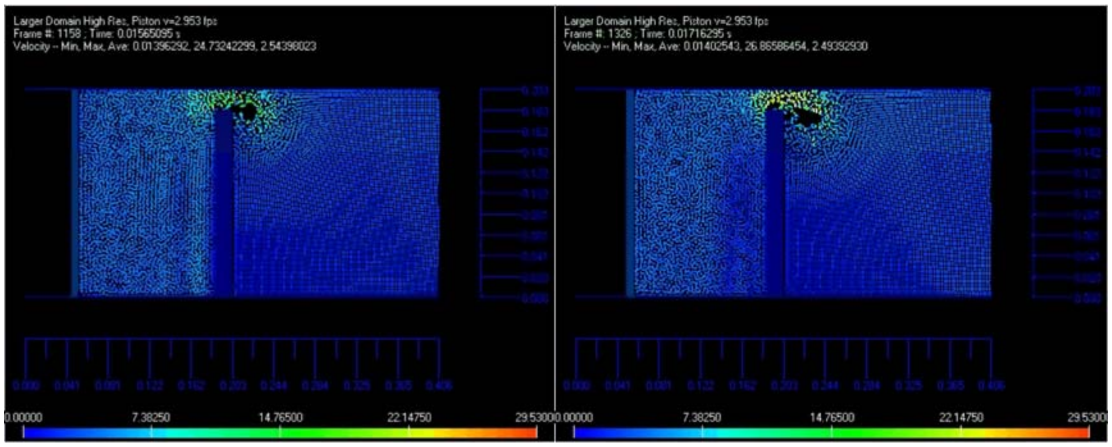


Figure 1.5: Formation of "cavities"

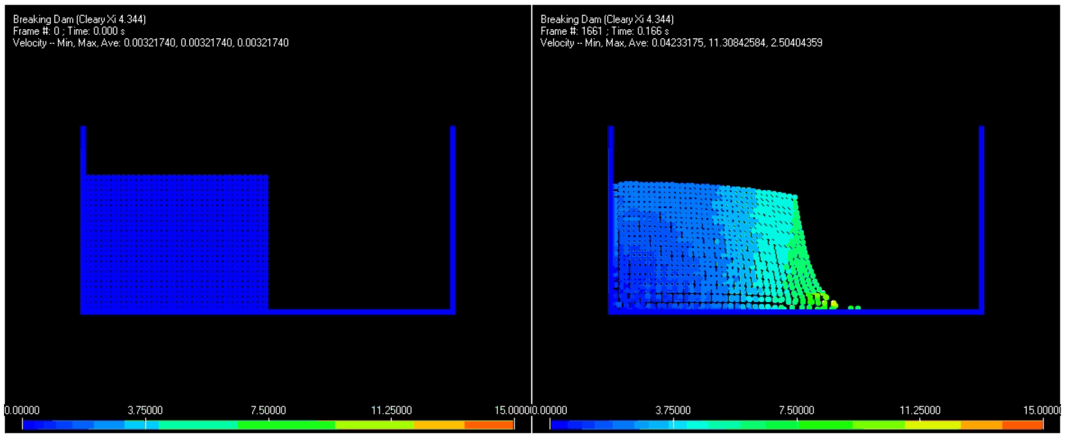
These figures show that the fluid speeds up as expected as it's forced through the gap. What was surprising with this simulation was the formation of "cavities" in the gap area that initiated at the leading edge of the gap and were convected through the gap into the wake. Moreover, these "cavities" seemed to exhibit some periodicity in that as one was shed into the wake another was formed soon after at the leading edge of the gap. I say these results are surprising not because I didn't think that such a flow situation could result in cavitation, but instead because I made no attempt to model cavitation in this version of the code. Initially, I wasn't sure what to make of these cavities and was thus inspired to investigate cavitating flow further using SPH or a derivative thereof (the DPM as it turns out).

After gaining much more experience using SPH, and DPM, I now realize that these cavities must be the result of centrifugal acceleration as the particles' velocities follow the curved path in the wake eddies. That centripetal acceleration tends to pull the particle away from the center of rotation resulting in the cavity. However, what we really have is a void in the flow simulation where there is no computational capacity since the computational nodes have evacuated the region. Therefore, we're unable to capture any flow information in these void spaces. This turns out to be a significant drawback of Lagrangian methods since certain flow characteristics of interest can result in the loss of resolution in the regions where resolution is most needed or desired. As you'll see later, the DPM is a hybrid method in that it can perform as either a Lagrangian method or an Eulerian method allowing one to choose the approach best suited for the problem at hand.

Breaking dam

The following series of images (Figures 1.6, 1.7, 1.8, and 1.9) show a few frames from a breaking dam simulation using this present program. The color scale represents particle speed with red representing higher speeds and blue representing low or zero speed.

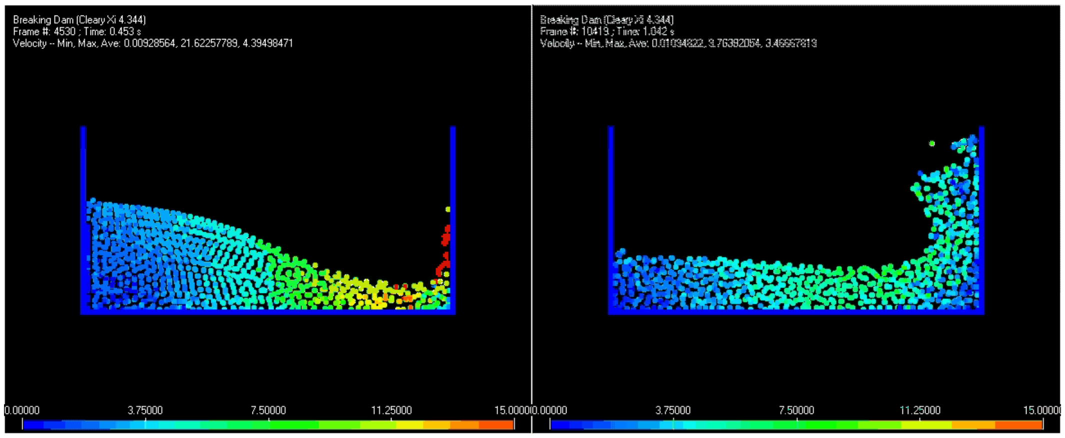
In the breaking dam simulation a column of fluid is initially retained by a vertical wall. At the start of the simulation that wall is removed and the fluid column is free to collapse in this case within a rectangular container. Figure 1.6 shows two time instances during this simulation. The frame on the left represents time zero – the instant the retaining wall is removed. The frame on the right represents a short period of time after the wall has been removed. Here the column is beginning its collapse whereby the base of the column advances to the right faster than top of the column as the fluid falls under the influence of gravity.


Breaking Dam (Cleary Xi 4.344)
Frame #: 0 ; Time: 0.000 s
Velocity -- Min, Max, Ave: 0.00321740, 0.00321740, 0.00321740

Breaking Dam (Cleary Xi 4.344)
Frame #: 1661 ; Time: 0.166 s
Velocity -- Min, Max, Ave: 0.04233175, 11.30842584, 2.50404359

Figure 1.6: Initial particle configuration and start of motion

Figure 1.7 shows two time instances where the falling column has reached the opposite end of the container and begins to splash up the container wall.


Breaking Dam (Cleary Xi 4.344)
Frame #: 4530 ; Time: 0.453 s
Velocity -- Min, Max, Ave: 0.00928564, 21.62257789, 4.39498471

Breaking Dam (Cleary Xi 4.344)
Frame #: 10419 ; Time: 1.042 s
Velocity -- Min, Max, Ave: 0.01094822, 3.76392054, 3.46667813

Figure 1.7: Fluid impacting container wall

As the fluid splashes up the wall it begins to curl back onto itself as illustrated in Figures 1.8 and 1.9. Figure 1.9 shows a secondary splash as the curling fluid impacts onto itself.

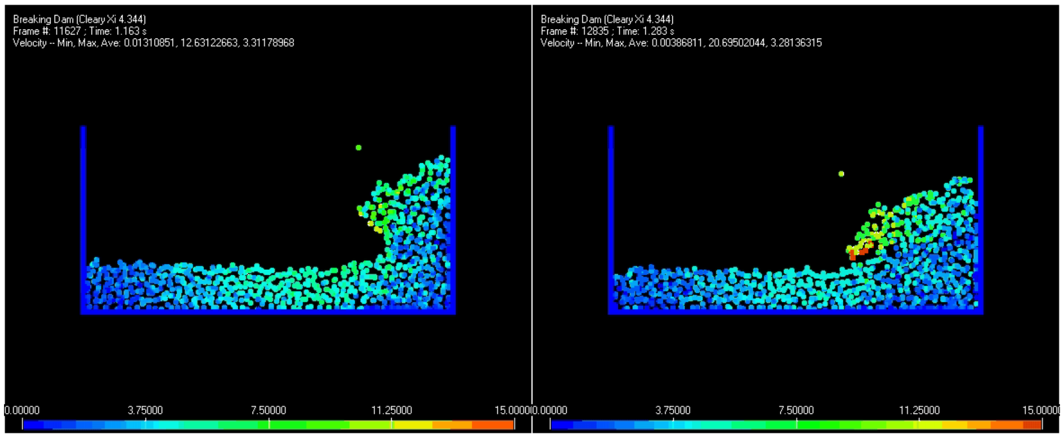

Breaking Dam (Cleary Xi 4.344)
Frame #: 11627 ; Time: 1.163 s
Velocity -- Min, Max, Ave: 0.01310851, 12.63122663, 3.31178968
0.00000
3.75000
7.50000
11.25000
15.00000
Breaking Dam (Cleary Xi 4.344)
Frame #: 12835 ; Time: 1.283 s
Velocity -- Min, Max, Ave: 0.00386811, 20.69502044, 3.28136315
0.00000
3.75000
7.50000
11.25000
15.00000

Figure 1.8: Fluid breaking back on itself

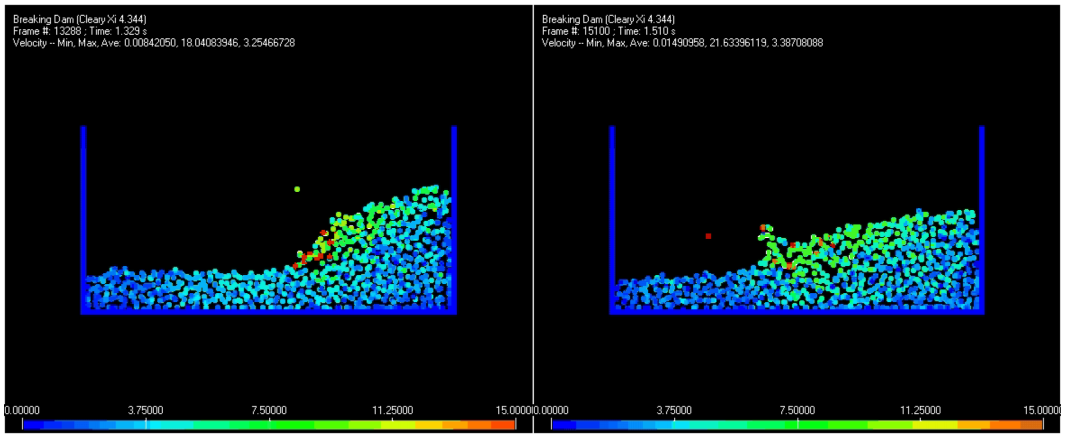

Breaking Dam (Cleary Xi 4.344)
Frame #: 13288 ; Time: 1.329 s
Velocity -- Min, Max, Ave: 0.00842050, 18.04083946, 3.25466728
0.00000
3.75000
7.50000
11.25000
15.00000
Breaking Dam (Cleary Xi 4.344)
Frame #: 15100 ; Time: 1.510 s
Velocity -- Min, Max, Ave: 0.01490958, 21.63396119, 3.38708088
0.00000
3.75000
7.50000
11.25000
15.00000

Figure 1.9: Initial secondary splash created by fluid-fluid impact

The results of this simulation show good qualitative agreement with those reported in the literature as well as the previously mentioned experimental results. On the other hand, these initial tests did indeed show that boundary particle representation along with repulsion force models can result in sever numerical instabilities. In some test runs, particles in the lower left of the container, where velocity should be low, experienced erratic motion due to extremely high repulsion forces. Subsequent particle-particle collisions exasperated the problem such that the region of instability grew and spread rapidly throughout the surrounding fluid until the entire simulation broke down. Careful tuning of such parameters as time step, smoothing length, artificial damping coefficients, and repulsion force coefficients was required to ensure a stable simulation. I should confess that this initial implementation was based on the most straightforward SPH techniques. Recent developments with the method, such as adaptive SPH, "remeshing," and methods to improve interpolation consistency near boundaries facilitate improved simulations that are more accurate, stable, and that do not require prohibitively small time steps.

Methods for Dealing with Cavity Flow

Aside from the motivation provided from my initial tests using SPH, cavity flows in and of themselves are an interesting phenomenon worthy of study. Cavitation is prevalent in a wide variety of fluid flow problems; everything from industrial process flows, to pumps and propellers, to high speed underwater projectiles, to space craft engines, to artificial heart pumps and valves involves cavitating flow. The phenomenon is widespread, and relevant. It's understanding is critical in applications where its detrimental effects must be mitigated or controlled. Cavitation, especially at high Reynolds numbers where turbulence is involved, is a complex phenomenon and is difficult to model accurately. Attempts to model cavitating flow have been undertaken for over one hundred years [32].

Classical methods for modeling fully developed cavity flow include potential flow methods that assume inviscid, irrotational and incompressible flow.[32] [34] For flow outside the thin boundary layer and for problems with well defined, easily predictable cavity detachment points these models yield reasonable results. They tend to break down when the detachment point is not well defined. In some problems cavitation does not occur on the body or at a sharp detachment point but instead occurs in the wake (in shed vortices for example).[36][37] Even for cleanly detached cavity flow, classical methods have difficulty modeling cavity closure as this area consists of turbulent flow patterns not easily handled using distributions of sources or boundary elements. [32]

Attempts to resolve cavity closure using potential flow methods have led to various so-called closure models. These models include simple models where the cavity is assumed to close at a neat, well defined point as well as more complicated models where the cavity closure forms a re-entrant jet. The single point closure approach is clearly unrealistic, and even the re-entrant jet model represents far more ordered flow patterns than what actually exist.[32]

Unsteadiness in the wake and cavity closure regions also present difficulties for classical methods. Other factors that present difficulties for any method, and which are gaining attention with newer methods, include effects such as surface tension, viscosity, turbulence, buoyancy (baroclinic vorticity generation) in closure regions, and compressibility of vapor or non-condensable gas and of the multi-phase fluid mixture.[41][52][54]

The authors of [40] discuss various computation approaches for dealing with cavitating flows. These methods include Eulerian, Lagrangian, and hybrid methods. The authors recognize, and this is backed up by other authors, that multiphase flow is very complex and computational methods still are not well developed to deal with many of the previously mentioned complexities inherent in cavitating flow.

Many modern methods use single fluid models where volume fractions and mixture density is evolved using transport equations, while other methods use multi-fluid models with transport equations suited to each component of the flow. In most cases, the methods employ some sort of mesh or computational grid. Such grids always present problems where sharp interfaces exist and resolution may be lost. Therefore many methods attempt to employ some form of interface tracking algorithm such as level sets or one-way coupled Lagrangian methods. In these cases, usually the non-cavitating velocity field is computed with a method such as a RANS code and then fronts (e.g., using implicit surfaces) or particles are evolved subject to the previously computed velocity field [53]. These methods ignore the effects of cavities on the velocity field.

Several methods, [48], [49] and [51] for example, incorporate bubble or non-condensable gases along with vaporization models and track bubble evolution using some form of the Rayleigh-Plesset equation. [51] discusses several limitations of this approach including numerical instability issues and inaccuracies when liquid-vapor density ratios are large (which often occurs in cavitating flow) and when pressure differences are large between inflow and outflow points in the computational domain. Further, results are reported to be very sensitive to initial, specified conditions and require the use of tuned relaxation factors to improve results.

Cavitation inception remains troublesome as theoretical and numerical predictions of cavitation inception often do not agree with experimentally obtained results. Researchers attribute this to several possible reasons including scaling effects and the role of cavitation nuclei. [32] and [63] through [67] report the strong influence on cavitation nuclei number density and size on cavitation inception. They further report that the standard engineering definition of cavitation inception occurring when the fluid pressure drops below the liquid vapor pressure leads to inaccurate predictions since effects of nuclei are ignored. Carefully controlled experiments indicate that liquids can sustain large tensions even when the pressure falls well below the vapor pressure without resulting in cavitation events. Moreover, flows with realistic nuclei densities have been shown experimentally to result in cavitation inception even when the fluid pressure has not fallen below the vapor pressure of the liquid. These results highlight the importance of cavitation nuclei on cavitation inception. Attempts have been made to model the cavitation nuclei in numerical simulations. However, difficulties have been encountered due to the different scales involved and the fact that many methods use a continuum model which is inherently inaccurate for simulating discrete nuclei and sharp free-surface like interfaces in cavitating flows. In general, the jury is still out as to which method/model best captures complicated cavitating flow phenomenon.

Several researchers [16] [30] have used SPH to model multi-phase flow in the context of astrophysics simulations, e.g. for such things as star formation. In these cases, they deal with compressible gases along with other physics such heat transfer, magnetic effects and gravity. More down to earth problems such as air entrainment in breaking waves [25] [31] [39], e.g., around ship bows or ocean waves breaking near shore, have been investigated using multi-fluid versions of SPH. In these latter cases, focus is on modeling two fluids such as air and water and not the multi-phase, with phase transition, phenomenon of cavitation. Some researchers [1] [2] have applied SPH to simulate high pressure die casting processes involving liquid to solid phase transitions. I feel that valuable techniques can be borrowed from all of these fields and applied to the simulation of cavitating flow problems.

At the start of this research I held the view that SPH, or a derivative thereof, offered several compelling advantages for application to cavitating flow problems; namely:

- The Lagrangian nature of SPH eliminates troublesome convective acceleration terms from the momentum equations

- SPH along with recent improvements for front tracking can handle sharp density gradients and material interfaces

- SPH readily facilitates incorporation of different physics/models, such as surface tension, chemical reactions, thermodynamics, etc.

- SPH can handle multiple fluids

- SPH naturally models compressible fluids

- Physical viscosity can be modeled in SPH

- Turbulence can be treated in SPH

- SPH models buoyancy

- SPH, due to its Lagrangian nature instead of a continuum approach, naturally handles particles and sharp interfaces facilitating the modeling of cavitation nuclei and free-surfaces

In general, I still hold this view with the exception of the Lagrangian nature mentioned above. My work has revealed that for cavity flows, at least the problem I studied, are better dealt with using an Eulerian approach. Fortunately, as mentioned earlier, the DPM works either way as will be discussed in greater detail later.

In spite of these advantages, SPH is not a silver bullet that can take care of any problem. SPH has difficulties dealing with boundaries (it was originally developed for boundless cosmological flows after all) and steep density gradients, and viscosity models still require quite a bit of tuning. Further, SPH often requires intermittent particle redistributions to ensure accuracy and resolution where desired (this is analogous to re-meshing in a finite element or boundary element approach).

Chapter 2

THE DISTRIBUTED POINTS METHOD (DPM)

The Distributed Points Method (DPM) is a meshless method for numerically solving differential equations, specifically, the Navier-Stokes equations in this study. The DPM is essentially a particle method similar to and inspired by the method of Smoothed Particle Hydrodynamics. Particle methods are attractive to use in the numerical modeling of complex fluid flows because the particles, which are computational nodes, are disconnected in the sense that there's no mesh or connectivity scheme to constrain node position. This means that for complex flows, such as free surface flows, there's no need to worry about issues such as mesh entanglement that cause serious problems in meshed based methods.

Splashing, fluid breakup and coalescence are easily handled by particle methods without the need for elaborate mesh management and re-meshing schemes. Moreover, considerable time is spent setting up and tuning meshes in meshed-based methods, whereas, particles are relatively easy to set up for meshless methods. That said, there are some problems that present difficulties for particle methods like SPH. For example, problems involving high density ratios generally require some form of particle redistribution intermittently during a simulation to ensure stability and reduce errors that occur when particles become disorganized. Density ratios on the order of 1000 to 1 are required for cavitating flow problems. Such density ratios are quite steep and care has been taken during development of the DPM to ensure the method can handle these density ratios.

An ability to handle high density ratios is just one of the goals I had in mind when developing the DPM. At the onset of this research, I wanted to develop a method that:

1. was meshless and could be used in either a Lagrangian or an Eulerian fashion.

2. could handle large density ratios, specifically density ratios of 1000-to-1.

3. would not have the truncation issue that SPH has near boundaries.

4. would be easy to extend to handle multiphase flow with phase transition.

5. would be scalable in terms of having the ability to relatively easily add more physics to the governing equations.

6. would be straightforward conceptually and relatively easy to program.

7. would be relatively easy to parallelize.

My rationale for the first goal listed is due to a recognition of the demonstrated flexibility and adeptness of SPH – a Lagrangian method – to handle a wide variety of complex flow problems. On the other hand, it's well known that specification and enforcement of boundary conditions is more straightforward in Eulerian methods. Further, Eulerian methods don't suffer from the issue of particles evacuating a specific location in the fluid domain due to dynamic action.

Large density ratios are present in the cavitation problems I'm interested in investigating; therefore, having the ability to cope with density ratios on the order of 1000-to-1 as stated in the second goal above is a requirement.

Regarding the third stated goal, while the SPH formulation for computing gradients is attractive in that gradients are simply analytically derived given the kernel approximation, the simplification made in SPH results in errors when applied near boundaries. There are several approaches for reducing the error in SPH, however, they add complexity to the overall formulation and implementation.

One of SPH's attractive features is that it's relatively easy to implement a wide variety of state equations describing the pressure-density relationship and to add new physics to the governing equations. Given my desire to adapt the new method to multiphase flow I wanted to retain this characteristic. Goals four and five stem from this desire.

As stated earlier in this dissertation, I found that SPH implementations were getting more complex and ad hoc as the method has been adapted to deal with different flows or overcome certain limitations with the method destroying one of the methods primary characteristics that has made it so popular in the first place, and that is its simplicity both conceptually and in implementation. As stated in goal six, I wanted to preserve a fundamental simplicity in the method I developed not only for selfish reasons as it would make my work easier but also to make the method readily accessible to students for pedagogical use and experimentation.

Finally, as stated in goal seven, I knew that leveraging computing clusters would be a requirement for performing complex, high resolution simulations. Therefore, I wanted a method, like SPH, that was relatively straightforward to parallelize.

Having already researched many state of the art computational fluid dynamics methods during my coursework, I knew that SPH was the method of choice for my purposes with the exception of the problems I've already discussed. Therefore, with SPH as a model and inspiration I began development of the new method by starting with the governing equations and figuring out the most straightforward way of computing the required derivatives of field values, after all, computation of these derivatives is the heart of any numerical scheme. My first decision was to retain the meshless nature of SPH along with its interpolation concept. However, I wanted to avoid transforming the governing equations and instead compute the required derivatives applying them directly in the original form of the governing equations.

After much trial and error and several failures, the solution I found was to develop shape functions representing the field values that required differentiation and to differentiate those shape functions analytically. In order to fit these shape functions to the underlying field values, I adopted the method of least-squares along with an SPH-like weighting function and nearest neighbor scheme. Basically, field values at any given particle are the weighted averages of values at that particle and its nearest neighbors. Further, the particles have finite support in that only neighbors in the immediate vicinity of any given particle contribute to the weighted average. This is essentially the SPH interpolation scheme. However, unlike SPH I don't use the derivative of the weighting function to approximate derivatives of field values; instead, I fit a shape function and differentiate the shape function. This approach is further discussed later in this chapter.

Another element borrowed from SPH was the use of an artificial compressibility approach rather than solving a pressure Poisson equation to ensure a divergence free velocity field. The artificial compressibility approach is somewhat more straightforward to implement, plus I wanted the flexibility to deviate from a purely incompressible flow solver since I anticipated compressibility being a factor in vapor filled cavity flows.

Governing Equations

The governing equations adopted in the DPM are the incompressible Navier-Stokes equations,

$$\rho\frac{D(u)}{Dt} = -\frac{\partial P}{\partial x} + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + g_x$$
$$\rho\frac{D(v)}{Dt} = -\frac{\partial P}{\partial y} + \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + g_y$$

All of the work carried out as part of this research has been in 2D; therefore, the above equations are the 2D Navier-Stokes equations. Extension of the method to 3D would involve adding another momentum conservation equation for the z-coordinate direction and selection of a suitable three dimensional shape function (to be discussed later). In these momentum conservation equations, ρ represents fluid density, *u* the fluid velocity in the x-direction, *v* the fluid velocity in the y-direction, *P* the pressure, μ the viscosity, g_x is a body force term in the x-direction, and g_y is a body force in the y-direction.

As in SPH the DPM uses an artificial compressibility approach to approximate incompressible flow and adopts the following form of the continuity equation,

$$\frac{d\rho}{dt} = -\left(u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y}\right) + \rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)$$

So far these three equations – two momentum conservation equations and the continuity equation – represent three equations with four unknowns *u*, *v*, ρ, and *P*. To close this equation system a suitable state equation relating pressure to density is introduced, namely,

$$P = c^2\frac{\rho_o}{\gamma}\left(\frac{\rho}{\rho_o}\right)^{\gamma}$$

Here, *c* is an artificial sound speed used to enforce incompressibility and γ is the specific heat ratio of the fluid (usually taken as 7 though in the phase change state equation discussed later in Chapter 5 it may differ from 7). This state equation is a standard barotropic state equation where pressure is an explicit function of density only. Temperature does not appear in this state equation although it certainly could in which case we would require an additional energy conservation equation to close the system. Some implementations of SPH take this approach when applied to problems where heat transfer is a significant part of the problem. I've ignored effects of temperature so far in this study. The state equation shown here is for single phase fluids such as water. Later in Chapter 5, I present a multiphase state equation developed for application of the DPM to cavitation.

As in SPH methods, DPM requires a suitable artificial sound speed, *c*, in order to approximate incompressible flow. The rule of thumb in SPH methods is that the resulting artificial Mach number should be less than or equal to 0.1 and I've adopted the same standard here; however, as part of a systematic sensitivity study to be discussed later I do vary the Mach number (see Chapter 4).

The momentum equations, shown earlier, are written using the standard substantial derivative operator. In the DPM, the momentum equations can be used in either Eulerian form, with convection terms, or Lagrangian form without convection terms. Expanding the substantial derivative term in the momentum conservation equations reveals the convective acceleration terms and the Eulerian form of these equations are thus as follows,

$$\rho\left(\frac{du}{dt}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right)=-\frac{\partial P}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+g_x$$
$$\rho\left(\frac{dv}{dt}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}\right)=-\frac{\partial P}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+g_y$$

In the Eulerian form, derivatives for convection terms are approximated using the weighted least squares method discussed later in this chapter. With this approach, the computational points, or nodes, distributed throughout the fluid domain remain fixed in space (unless a movable boundary is simulated). Each point stores the fluid properties relevant to that point in the fluid domain at any given instant in time. Fluid properties in essence pass through the points as illustrated conceptually in Figure 2.1.

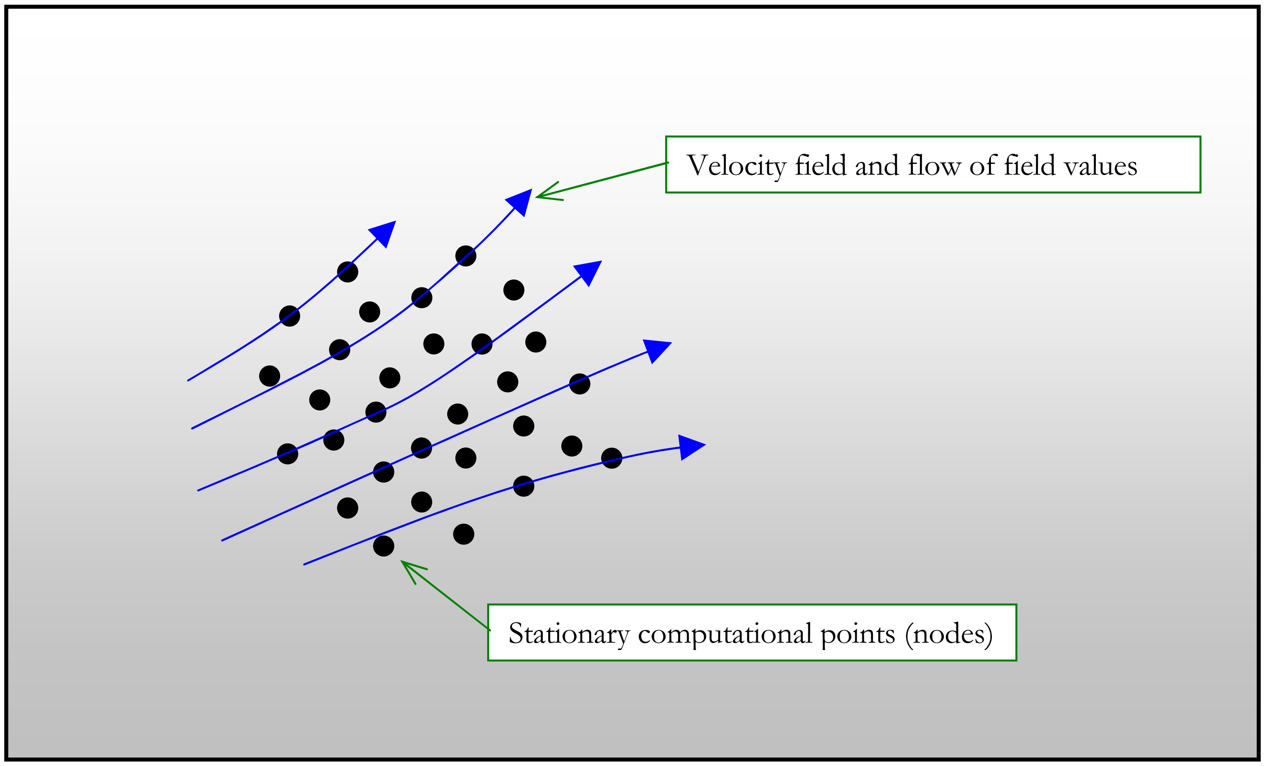

Velocity field and flow of field values
Stationary computational points (nodes)

Figure 2.1: Illustration of Eulerian Approach

In the Lagrangian form there are no convection terms to deal with, which means that fluid properties will not be advected throughout the flow field as in the Eulerian case. However, in the Lagrangian case, the computational points themselves are advected in an manner similar to that seen in the SPH method. Figure 2.2 illustrates this concept.

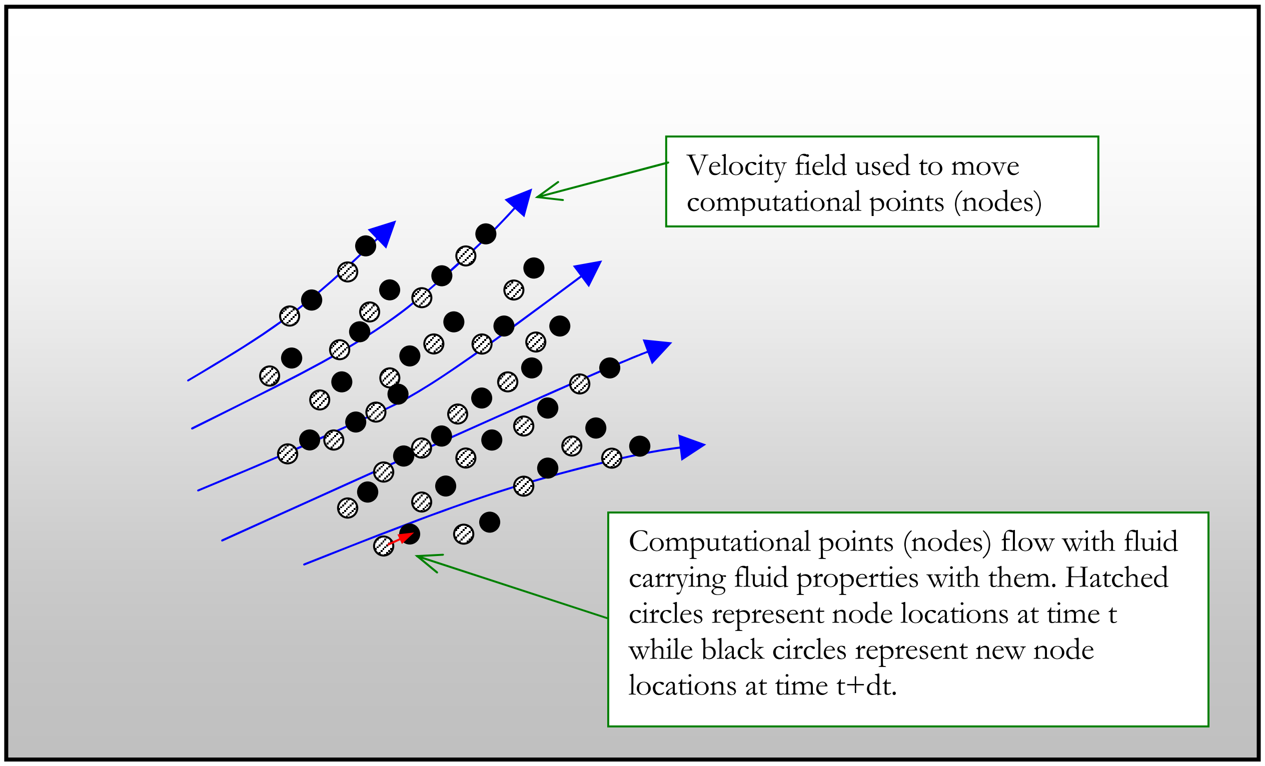

Velocity field used to move computational points (nodes)
Computational points (nodes) flow with fluid carrying fluid properties with them. Hatched circles represent node locations at time t while black circles represent new node locations at time t+dt.

Figure 2.2: Illustration of Lagrangian Approach

The Lagrangian form of the momentum equations are simply,

$$\rho \frac{du}{dt} = -\frac{\partial P}{\partial x} + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + g_x$$

$$\rho \frac{dv}{dt} = -\frac{\partial P}{\partial y} + \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + g_y$$

When the DPM uses this form, the points distributed throughout the fluid domain move with the fluid carrying fluid properties along with them. The points move by virtue of the velocity field they themselves represent and numerically it's a simple matter of integrating the velocity field to compute displacements for each point in the fluid domain.

Both the Lagrangian and Eulerian approaches have been used in this research and both have their advantages and disadvantages. For example, inflow and outflow boundary conditions are easier to handle using an Eulerian approach; however, the Lagrangian approach eliminates the need to deal with convective terms altogether. The DPM is flexible enough to handle both approaches affording options when modeling certain flow types. For example, free surface flows where splashing is important may be better handled using the Lagrangian approach, whereas the 2D cylinder flow problems discussed later in Chapter 4 are better handled using an Eulerian approach.

In the distributed points method, the governing fluid equations are not transformed for some discretization scheme but instead are used directly as shown. This is achieved through the use of suitable shape functions that represent fluid field values such as velocity and direct analytic differentiation of those shape functions. The shape functions are discussed next.

Gradient Computation

A key feature of the DPM is the combination of an interpolating kernel in the sense of SPH with least squares approximations for field values in order to compute spatial derivatives in the governing equations.

In SPH, interpolation kernels are used to derive an approximation of the gradient of field values in terms of the gradient of the interpolation kernel. This is a well known approximation in the SPH literature. It is also well known that such an approximation for gradients possesses several deficiencies when computing gradients near boundaries. In the DPM we use interpolation kernels in a somewhat different manner.

Given a set of dispersed points with spatial coordinates and associated field values, we can interpolate the field value of interest at any coordinate using an interpolation kernel as follows,

$$A(x_i) = \int A(x_k) W(x_i, x_k) dx$$

where W is a weight function that gives more weight to points, x_k, that are closer to the point x_i. This concept is illustrated in Figure 2.3.

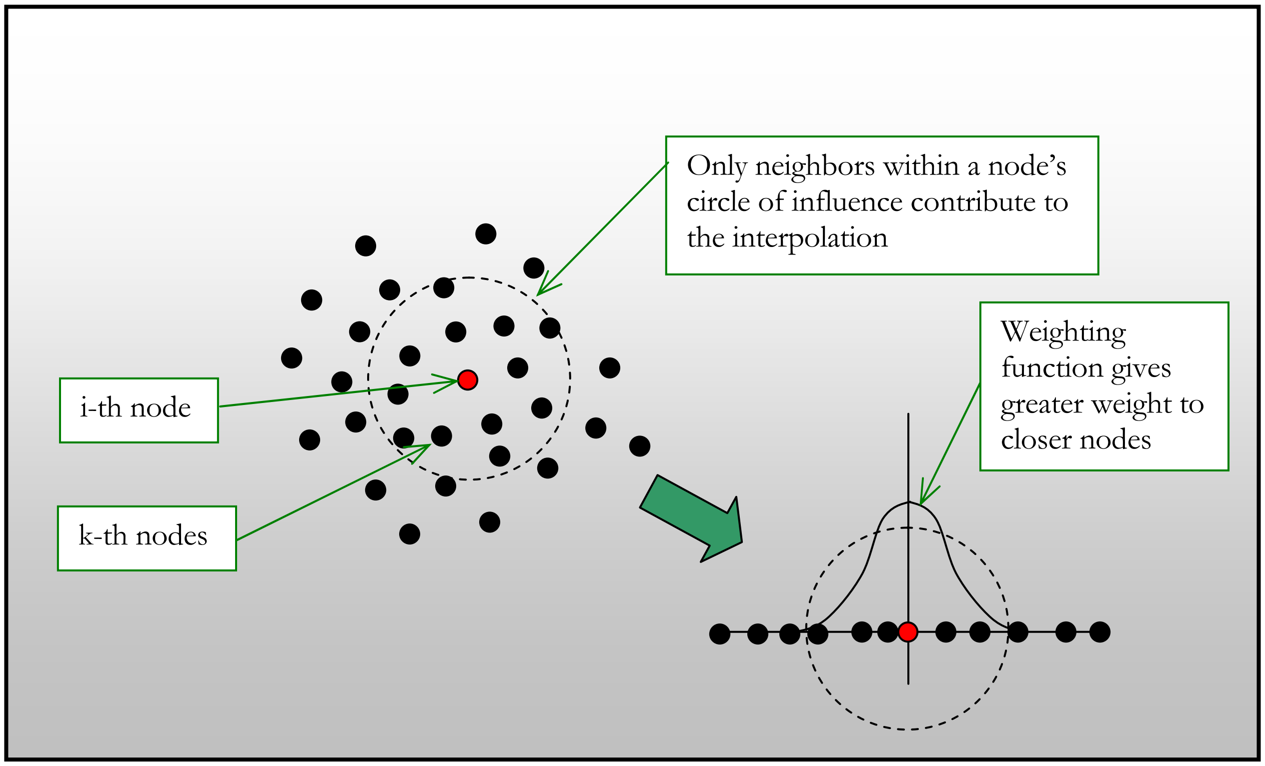

Only neighbors within a node's circle of influence contribute to the interpolation
Weighting function gives greater weight to closer nodes
i-th node
k-th nodes

Figure 2.3: Illustration of weighted interpolation

It should be noted that *A(x)* and *W(x)* are functions of the vector coordinates of each node such that in two dimensions *A* is a function of both *x* and *y*, as is *W*. In the case of a discrete set of unconnected points we have,

$$A(x_i) = \sum A(x_k) W(x_i, x_k)$$

over all *k* points.

This is the same interpolation formula forming the basis of SPH. In practice, this formula is essentially a weighted average of field quantities surrounding the point of interest. The important element to emphasize here is that values closer to the point of interest are given more importance – more weight – than values further from the point of interest.

In DPM, these weights are also used in another way. Specifically, the DPM uses a weighted least squares method to derive coefficients for shape functions representing field values where these shape functions are then differentiated analytically to compute spatial derivatives. This is where DPM starts to differ from SPH. As discussed earlier, SPH uses the gradient of the weight function to approximate derivatives and the governing equations are transformed to accommodate this approximation. Since in the DPM derivatives are computed for analytic shape functions directly representing field values, we can apply the governing equations directly without transformation.

In the DPM we assume that field values at any point can be approximated by a function of the form,

$$f = \sum a_n \varphi_n$$

The a_n coefficients are determined in the usual least squares manner by minimization of the weighted residual,

$$\min\left\lfloor \sum (F_k - f_k)^2 W(x_i - x_k) \right\rfloor$$

considering all *k* points in the support domain of the point, *i*, under consideration. *F* is the actual field value at the point considered and *f* is the approximated field value. Depending on the form of the shape function, there are several approaches that can be taken to find the a_n coefficients. For example, if the field values are locally approximated by a linear function (for two-dimensional problems where field values are functions of the x- and y-coordinates),

$$f = a_o + a_1 x + a_2 y$$

then the normal equations can be derived relatively easily and the coefficients solved for directly. For other functions, like the bi-quadratic function (again for two-dimensional problems),

$$f = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2$$

the normal equations are cumbersome to solve analytically (later in this chapter I present the normal equations for this shape function). An alternate approach is to use direct matrix inversion techniques though they are more computationally expensive. Numerical tests proved the bi-quadratic shape functions more stable and better capable of handling steep gradients, such as density gradients that appear in cavitating flow. Moreover, second derivatives can be computed easily for diffusive terms in the governing equations rather than using repeated linear approximations which result in greater approximation error.

Once the least squares coefficients have been determined for all field values – pressure, velocity, and density – at each node, spatial derivatives are computed by direct differentiation of the shape function. For example, first derivatives in *x* are,

$$\frac{\partial f}{\partial x} = a_1 + 2a_3 x + a_4 y$$

With proper centering of all coordinates about the x_i coordinate while performing the weighted least squares fit all terms involving *x* or *y* drop out, yielding simply,

$$\frac{\partial f}{\partial x} = a_1$$

Similarly, second derivatives reduce to the form,

$$\frac{\partial^2 f}{\partial x^2} = 2a_3$$

The challenge is to perform these least squares approximations efficiently since they must be performed for every node in the simulation for each time step. To put that in perspective, if you had 1000 nodes in a simulation and were time stepping every millisecond, you'd have to perform the least squares fit 1 million times for every second of simulation. Clearly there's some incentive to optimize this aspect of the method upon implementation, thus I've incorporated several optimizations in the code so that reasonable computation times can be achieved. For example, all of the simulations discussed in this paper were performed on a typical, scalar desktop computer with a Pentium III processor with simulation times on the order of a few hours for moderate resolution tests and only minutes for lower resolution tests. (High resolution tests were conducted with run times on the order of days on the same scalar computer.)

Least Squares Fit

Before settling on using the bi-quadratic shape function, with a direct solution to the normal equations, I tried several different approaches. These approaches include:

1. Use standard least-squares algorithms to fit bi-quadratic shape functions.

2. Derive the bi-quadratic normal equations and solve directly.

3. Derive a plane equation shape function for use when computing pressure and density gradients.

4. Derive a two-part shape function using quadratic polynomials.

5. Derive a two-part shape function using linear polynomials for computing pressure and density gradients.

My primary incentive for trying these five different approaches was to find the approach that yielded the most efficient computation time. As it turns out the best approach, and the one used for the simulations presented herein, is a combination of two of the above approaches. Bi-quadratic shape functions solved in closed form were used for velocity gradients where second derivatives are required. Plane equation shape functions solved in closed form were used for density and pressure gradients where only first derivatives are required. Each approach is discussed in greater detail in the following paragraphs.

Standard Least Squares Algorithm

The easiest approach to fitting the bi-quadratic shape function was simply to use a canned non-linear least squares fitting algorithm. I chose to implement the version described in Reference [72], Numerical Recipes in C Second Edition, which worked quite well in terms of fitting the bi-quadratic shape function to field values around any given node. The routines return the function coefficients allowing computation of the required derivatives. The problem with this approach is that it is very slow. To fit each shape function, a six-by-six matrix must be inverted. That's a matrix inversion for every computational node for each time step. It seemed to me that a quicker approach would be to solve the least squares normal equations by hand, analytically, and program the solutions for the fit coefficients avoiding any sort of runtime matrix inversion. This approach is discussed next.

Bi-quadratic Shape Function Solved Directly

Applying the method of least squares to find the six coefficients of the bi-quadratic shape function shown earlier requires finding the simultaneous solution to the six normal equations representing the minimization of weighted, squared residuals. The normal equations for the bi-quadratic shape function are found by taking the derivative of the sum of weighted, squared residuals expression shown earlier for each of the six coefficients, a_n, and equating the resulting expressions to zero. For example, let

$$q = \sum (F_k - f_k)^2 W(x_i - x_k)$$

which is the expression we want to minimize. F_k represents the actual field value at the *k*th nearest neighbor to node *i* under consideration, while f_k represents the approximated field value at the *k*th nearest neighbor using the bi-quadratic shape function. *W* represents the weight value for the *k*th nearest neighbor as a function of the distance from the *k*th neighbor to the *i*th node under consideration. Letting *z* be the value of F_k, *w* be the weight for *k*th neighbor, and substituting the bi-quadratic shape function yields the following expression for *q*,

$$q = \sum \left[\left(z - \left(a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 \right) \right)^2 w \right]$$

Here the summation is over *n* nearest neighbors to node *i*, including node *i*, and the *k* subscripts have been dropped for simplicity.

To get the first of the six normal equations you take the partial derivative of *q* with respect to the first coefficient, a_0 as follows,

$$\frac{\partial q}{\partial a_0} = \sum \left[-2 \left(z - \left(a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 \right) \right) w \right]$$

Setting the derivative of q with respect to a_0 equal to zero, rearranging, and distributing the summation yields the following normal equation,

$$\sum zw = a_0\sum w + a_1\sum xw + a_2\sum yw + a_3\sum wx^2 + a_4\sum xyw + a_5\sum wy^2$$

Following a similar procedure for each of the five remaining a_n coefficients yields five additional normal equations as follows,

$$\begin{aligned}
\sum xzw &= a_0\sum xw + a_1\sum x^2w + a_2\sum xyw + a_3\sum wx^3 + a_4\sum x^2yw + a_5\sum wxy^2\\
\sum yzw &= a_0\sum yw + a_1\sum xyw + a_2\sum y^2w + a_3\sum wx^2y + a_4\sum xy^2w + a_5\sum wy^3\\
\sum x^2zw &= a_0\sum x^2w + a_1\sum x^3w + a_2\sum x^2yw + a_3\sum wx^4 + a_4\sum x^3yw + a_5\sum wx^2y^2\\
\sum xyzw &= a_0\sum xyw + a_1\sum x^2yw + a_2\sum xy^2w + a_3\sum wx^3y + a_4\sum x^2y^2w + a_5\sum wxy^3\\
\sum y^2zw &= a_0\sum y^2w + a_1\sum xy^2w + a_2\sum y^3w + a_3\sum wx^2y^2 + a_4\sum y^3xw + a_5\sum wy^4
\end{aligned}$$

These six normal equations represent a system of six equations with six unknowns, the a_n coefficients. Matrix inversion algorithms could be used to solve this system, however, that is essentially the approach taken when using standard least squares algorithms as discussed earlier. A speedier approach from a computational point of view would be to solve this system manually deriving a set of equations representing the coefficient values. Such equations are readily available in published literature for the standard solution to the least squares best fit straight line of the form $y = mx + b$, but not so for the least squares best fit bi-quadratic function. Therefore, I manually solved this system using Gaussian elimination with back-substitution which yielded a set of six equations representing the values of each of the a_n coefficients. These equations are,

$$
\begin{aligned}
a_5 &= \frac{\beta_{67}}{\beta_{66}} \\
a_4 &= \frac{\left(\theta_{57} - a_5\theta_{56}\right)}{\theta_{55}} \\
a_3 &= \frac{\left(\gamma_{47} - a_4\gamma_{45} - a_5\gamma_{46}\right)}{\gamma_{44}} \\
a_2 &= \frac{\left(\phi_{37} - a_3\phi_{34} - a_4\phi_{35} - a_5\phi_{36}\right)}{\phi_{33}} \\
a_1 &= \frac{\left(\phi_{27} - a_2\phi_{23} - a_3\phi_{24} - a_4\phi_{25} - a_5\phi_{26}\right)}{\phi_{22}} \\
a_0 &= \frac{\left(\phi_{17} - a_1\phi_{12} - a_2\phi_{13} - a_3\phi_{14} - a_4\phi_{15} - a_5\phi_{16}\right)}{\phi_{11}}
\end{aligned}
$$

The β terms in these equations are,

$$
\begin{aligned}
\beta_{66} &= \theta_{55}\theta_{66} - \theta_{65}\theta_{56} \\
\beta_{67} &= \theta_{55}\theta_{67} - \theta_{65}\theta_{57}
\end{aligned}
$$

The θ terms are,

$$
\begin{aligned}
&\theta_{jk} = \gamma_{44}\gamma_{jk} - \gamma_{j4}\gamma_{4k} \\
&for, \\
&j = 5,6 \\
&k = 5,6,7
\end{aligned}
$$

The γ terms in these equations are,

$$
\begin{aligned}
&\gamma_{jk} = \phi_{33}\phi_{jk} - \phi_{j3}\phi_{3k} \\
&for, \\
&j = 4,5,6 \\
&k = 4,5,6,7
\end{aligned}
$$

And, the ϕ terms in these equations are,

$\phi_{00} = N$
$\phi_{01} = A$
$\phi_{02} = B$
$\phi_{03} = T$
$\phi_{04} = D$
$\phi_{05} = I$
$\phi_{06} = C$

$\phi_{10} = 0$
$\phi_{11} = TN - AA$
$\phi_{12} = DN - CB$
$\phi_{13} = HN - AT$
$\phi_{14} = EN - AD$
$\phi_{15} = FN - AI$
$\phi_{16} = ON - AC$

$\phi_{20} = 0$
$\phi_{21} = 0$
$\phi_{22} = (IN - BB)(TN - AA) - (DN - BA)(DN - AB)$
$\phi_{23} = (EN - BT)(TN - AA) - (DN - BA)(HN - AT)$
$\phi_{24} = (FN - BD)(TN - AA) - (DN - BA)(EN - AD)$
$\phi_{25} = (UN - BI)(TN - AA) - (DN - BA)(FN - AI)$
$\phi_{26} = (PN - BC)(TN - AA) - (DN - BA)(ON - AC)$

$\phi_{30} = 0$
$\phi_{31} = 0$
$\phi_{32} = (TN - AA)(EN - TB) - (HN - TA)(DN - AB)$
$\phi_{33} = (TN - AA)(LN - TT) - (HN - TA)(HN - AT)$
$\phi_{34} = (TN - AA)(KN - TD) - (HN - TA)(EN - AD)$
$\phi_{35} = (TN - AA)(GN - TI) - (HN - TA)(FN - AI)$
$\phi_{36} = (TN - AA)(RN - TC) - (HN - TA)(ON - AC)$

$$\phi_{40} = 0$$
$$\phi_{41} = 0$$
$$\phi_{42} = (TN - AA)(FN - DB) - (EN - DA)(DN - AB)$$
$$\phi_{43} = (TN - AA)(KN - DT) - (EN - DA)(HN - AT)$$
$$\phi_{44} = (TN - AA)(GN - DD) - (EN - DA)(EN - AD)$$
$$\phi_{45} = (TN - AA)(JN - DI) - (EN - DA)(FN - AI)$$
$$\phi_{46} = (TN - AA)(QN - DC) - (EN - DA)(ON - AC)$$

$$\phi_{50} = 0$$
$$\phi_{51} = 0$$
$$\phi_{52} = (TN - AA)(UN - IB) - (FN - IA)(DN - AB)$$
$$\phi_{53} = (TN - AA)(GN - IT) - (FN - IA)(HN - AT)$$
$$\phi_{54} = (TN - AA)(JN - ID) - (FN - IA)(EN - AD)$$
$$\phi_{55} = (TN - AA)(MN - II) - (FN - IA)(FN - AI)$$
$$\phi_{56} = (TN - AA)(SN - IC) - (FN - IA)(ON - AC)$$

Finally, the coefficients appearing on the right sides of all the ϕ terms are,

$$A = \sum_{k=1}^{n} w_k x_k$$
$$B = \sum_{k=1}^{n} w_k y_k$$
$$C = \sum_{k=1}^{n} w_k z_k$$
$$D = \sum_{k=1}^{n} w_k x_k y_k$$
$$E = \sum_{k=1}^{n} w_k x_k^2 y_k$$
$$F = \sum_{k=1}^{n} w_k x_k y_k^2$$

$$G = \sum_{k=1}^{n} w_k x_K^2 y_k^2$$

$$H = \sum_{k=1}^{n} w_k x_k^3$$

$$I = \sum_{k=1}^{n} w_k y_k^2$$

$$J = \sum_{k=1}^{n} w_k x_k y_k^3$$

$$K = \sum_{k=1}^{n} w_k x_k^3 y_k$$

$$L = \sum_{k=1}^{n} w_k x_k^4$$

$$M = \sum_{k=1}^{n} w_k y_k^4$$

$$N = \sum_{k=1}^{n} w_k$$

$$O = \sum_{k=1}^{n} w_k x_k z_k$$

$$P = \sum_{k=1}^{n} w_k y_k z_k$$

$$Q = \sum_{k=1}^{n} w_k x_k y_k z_k$$

$$R = \sum_{k=1}^{n} w_k x_k^2 z_k$$

$$S = \sum_{k=1}^{n} w_k y_k^2 z_k$$

$$T = \sum_{k=1}^{n} w_k x_k^2$$

$$U = \sum_{k=1}^{n} w_k y_k^3$$

Using these equations to find the unknown coefficients is dramatically faster computationally than using the standard least squares algorithms discussed earlier. While these equations may look cumbersome they are relatively straightforward to program.

As discussed earlier, with proper centering of coordinates, partial derivatives of field values approximated with the bi-quadratic function are simply:

$$\frac{\partial f}{\partial x} = a_1$$

$$\frac{\partial^2 f}{\partial x^2} = 2a_3$$

$$\frac{\partial f}{\partial y} = a_2$$

$$\frac{\partial^2 f}{\partial y^2} = 2a_5$$

This formulation is useful for approximating the velocity field where first and second derivatives are required as seen in the governing Navier-Stokes equations. For pressure and density, however, second derivatives are not required and a simpler formulation may be used.

Plane Equation Shape Function

The plane equation,

$$f = a_o + a_1 x + a_2 y$$

is used to approximate pressure and density field values. Normal equations for a plane function approximation can be derived by applying the same least squared techniques demonstrated earlier for the bi-quadratic function. The resulting a_n coefficients (shown here without derivation) are as follows:

$$a_0 = \frac{(D - Bb - Cc)}{A}$$
$$a_1 = \frac{N}{M} - \frac{Gc}{M} + \frac{BCc}{AM}$$
$$a_2 = \frac{\left(L - \frac{CD}{A} + \frac{CBN}{AM} - \frac{GN}{M}\right)}{\left(PC - \frac{G^2}{M} + \frac{BCG}{AM} + K\right)}$$

where,

$$A = \sum_{k=1}^{n} w_k$$
$$B = \sum_{k=1}^{n} w_k x_k$$
$$C = \sum_{k=1}^{n} w_k y_k$$
$$D = \sum_{k=1}^{n} w_k f_k$$
$$F = \sum_{k=1}^{n} w_k x_k^2$$

$$G = \sum_{k=1}^{n} w_k x_K y_k$$
$$H = \sum_{k=1}^{n} w_k x_k f_k$$
$$K = \sum_{k=1}^{n} w_k y_k^2$$
$$L = \sum_{k=1}^{n} w_k x_k f_k$$

and,

$$M = F - \frac{B^2}{A}$$
$$N = H - \frac{BD}{A}$$
$$P = \frac{BG}{AM} - \frac{B^2 C}{A^2 M} - \frac{C}{A}$$

Now partial derivatives reduce to,

$$\frac{\partial f}{\partial x} = a_1$$

$$\frac{\partial f}{\partial y} = a_2$$

In principle, the bi-quadratic function could also be used to approximate pressure and density derivatives; however, it involves more computations making the plane equation approach more efficient. Moreover, numerical tests using the bi-quadric equation to approximate pressure resulted in relatively noisy pressure fields.

Two-part Polynomial Shape Functions

An alternative approach to representing velocity field values and computing their derivatives is to use a two-part quadratic polynomial shape function. This approach utilizes a projection technique to simplify deriving normal equations which can then be used to compute derivatives. The projection works by compressing all points within a given point's support domain onto planes converting a three dimensional surface approximation into a pair of two dimensional curve approximations that are used only for approximating derivatives.

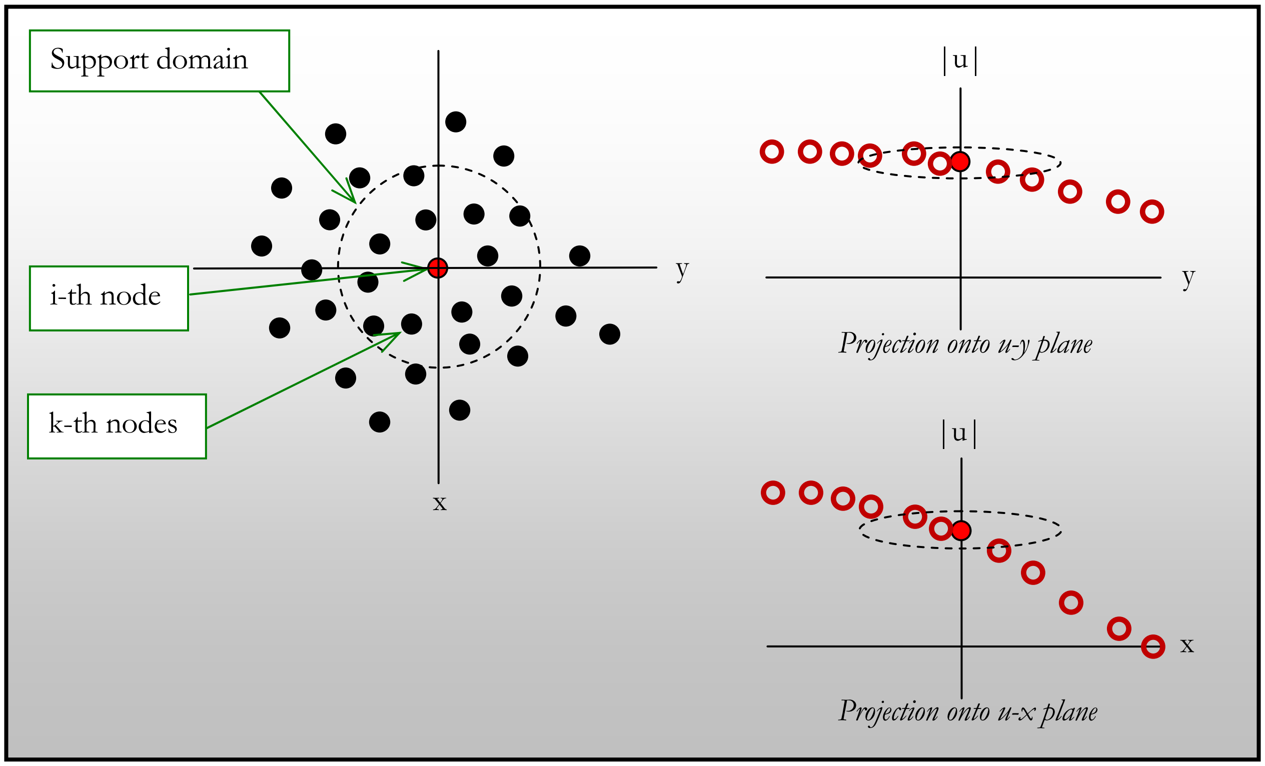


Figure 2.4: Two-part shape functions

For example, referring to Figure 2.4, we're interested in the derivatives of the field values, u_i, at the point represented by node *i*. All k nodes store the field value, u_k, at their respective locations. If you graph *u* as a function with respect to position *x* and *y*, you'd get a surface. We're interested in the derivatives of u_i with respect to *x* and u_i with respect to *y*. Graphically speaking we can compress all nodal values onto the *u-x* plane around the *i*th point of interest. We further compute the relative coordinates of each *k*th point with respect to the *i*th point. The resulting plot shows how *u* changes with respect to *x*. This two dimensional graph can then be fit with a function using a weighted least squares method as discussed herein. Now, the weights used to weigh the contributions of the *k* point values must be a function of distance in the *x*, *y* plane from the *k*th point to the *i*th point.

Repeating this process and compressing the points onto the *u-y* plane will result in another graph for which we can fit another function. The first function can be differentiated to approximate the derivative of *u* with respect to *x*, while the later function can be differentiated to approximate the derivative of *u* with respect to *y*.

Using this approach for the velocity field would require at least a quadratic interpolation function allowing the computation of second derivatives. For pressure and density fields, linear shape functions can be used.

This approach generally works fairly well. Numerical tests show relatively good computation efficiency and the approach is straightforward to program for two dimensional flow simulations. The drawback with this approach is that it's more cumbersome to implement for three dimensional flow simulations. Given my desire to extend this method to three dimensional problems, I've opted to retain the bi-quadratic and plane equation shape function approaches discussed earlier.

Weighting Function

The least squares approximation just described relies on a weighted least squares approach where nodes further from the given node under consideration, the *i*th node, are given less importance – *weight* – while performing the least squares calculations. This approach mimics physical reality in that activity at a point in the fluid is less and less influenced by activities further and further away from that point.

Chapter 1 describes how weighting functions are used in SPH. The most common weight function used in SPH is the cubic spline approximation to a Gaussian weight function as shown in Chapter 1. Many different forms of the weight function may be used; however, SPH does require the weight function to possess certain characteristics, namely: that the function is normalized; that it is differentiable, analytically, to the degree required in the particular SPH formulation; that it has a smooth second derivative; that the function has compact support; that the function is even and positive; that the function approaches the Dirac delta function as the smoothing length goes to zero.[24]

These same conditions generally apply in the DPM method except they do not necessarily have to be applied as strictly. For example, in the DPM the weight function need not be an analytic function at all since it isn't used in the same manner as in SPH where its analytic derivative is used directly. In the DPM, the weight function need not be normalized either. Moreover, the weighting function need not be even and, in fact, it is possible to construct a weight function that is skewed so as to give more weight to points on one side of the node under consideration versus those on the other side. This approach may have some usefulness by giving rigid boundary nodes on one side of a node more importance than the fluid nodes on the other side so as to more strictly enforce rigid boundary conditions.

The difference in the way SPH and the DPM use weight functions represents a fundamental difference between the two methods. In SPH spatial derivatives are computed using the analytic derivatives of the weight (smoothing function) as discussed earlier. In this regard the selected weight function is of critical importance in SPH. In the DPM method, weights are essentially inverses of individual standard deviations of the value at each node in a weighted least squares sense. Weights are used to quantify the relative importance of nodal values in the interpolation scheme and beyond that capacity the weights are not used.

Choices abound as to the form of the weight function that may be used in the DPM method. The most generally applicable form is that of a Gaussian function. Many approximate forms of the Gaussian have been implemented in SPH; the cubic spline weight function shown in Chapter 1 is one such example. I employed several weight functions in the DPM in an effort to test results sensitivity to the weight function. These functions include the cubic spline function, the Gaussian function, and a linear weight function. The cubic spline function was discussed in Chapter 1 and it's implementation in the DPM is straightforward.

The Gaussian function used is of the form,

$$w(|r|) = \frac{k}{ch\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{|r|}{ch}\right)^2}$$

where *h* is radius of support, $|r|$ is the distance from the node under consideration to the k$^{\text{th}}$ node, *c* is used to control the width of the Gaussian function, and *k* is generally selected to normalize the function. The Gaussian function is plotted in Figure 2.6

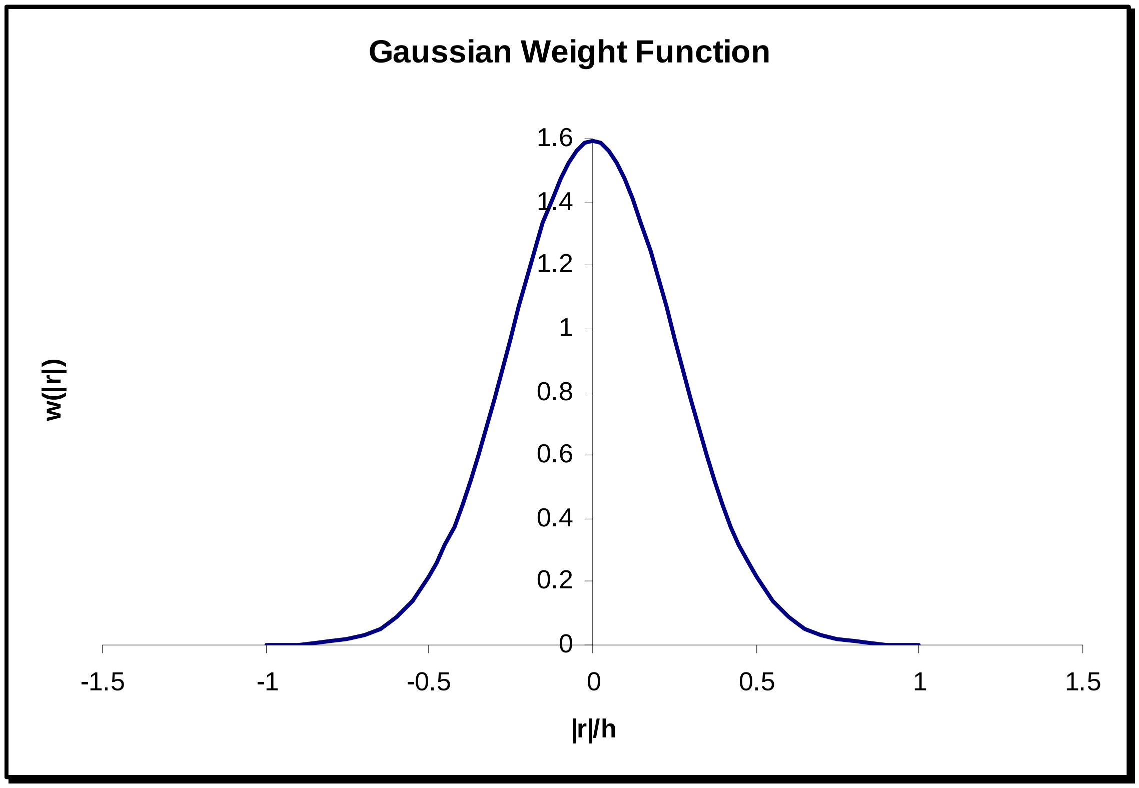


Figure 2.6: Gaussian function

The Gaussian does not possess compact support in the sense that as |r| increases, w(|r|) never actually reaches zero. This is one reason why the cubic spline and other approximations to the Gaussian function are commonly used in SPH. In those approximations you can guarantee that nodes outside the support radius will not contribute to derivatives of field values associated with the particular node under consideration.

A simpler and more computationally efficient weight function used in the DPM is the linear weight function of the form,

$$w(|r|) = \begin{cases} 1 - \dfrac{|r|}{h}; & for|r| \leq h \\ 0; & for|r| > h \end{cases}$$

This function is plotted in Figure 2.7.

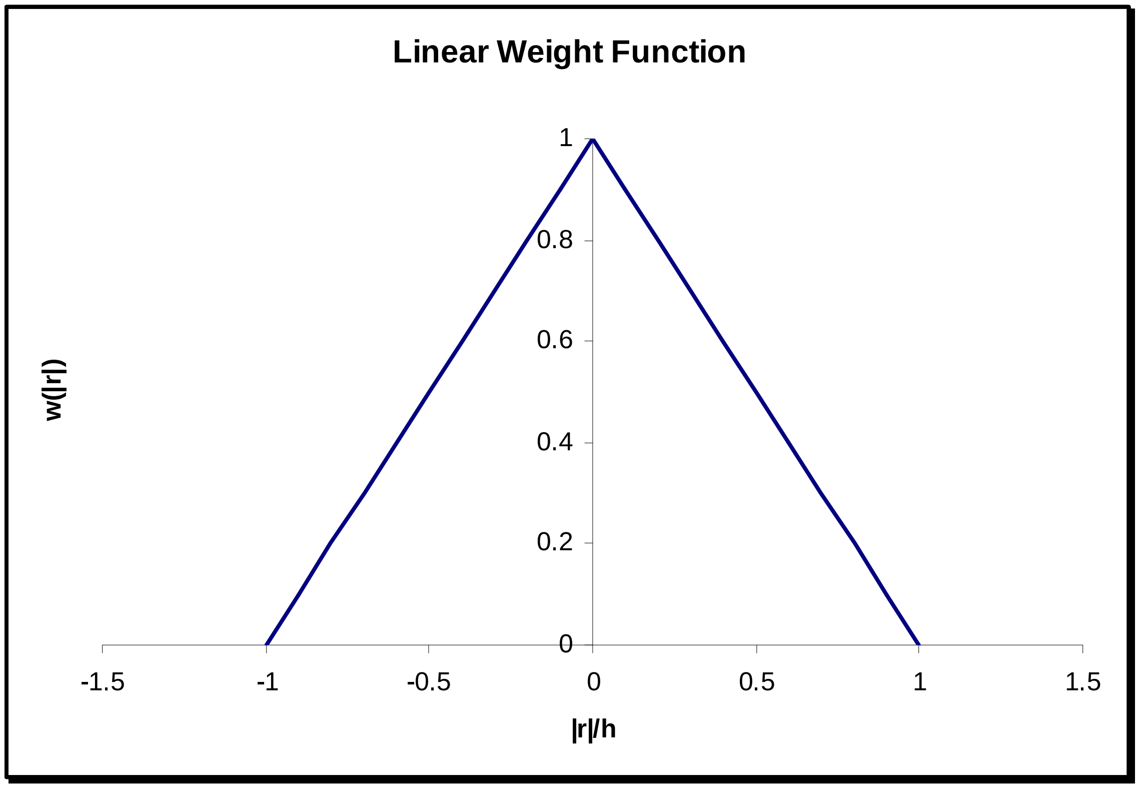


Figure 2.7: Linear weight function

Numerical tests show that this linear weight function, in the DPM, works just as well as the Gaussian function while performing the weight least squares interpolation function approximations. Given the improved computational efficiency offered by the linear function it is the weight function of choice.

While the precise shape of the weight functions used in the DPM are not as critical as they are in SPH, the peakedness, or width, of the weight function in the DPM has important influences on the results. If you think of the weight function as a filter, the wider it is, the greater it's filter effect in much the same way as a Gaussian low pass filter. Wider weight functions tend to smooth the field value interpolation, whereas, narrower weight functions have the opposite effect and thus preserve some higher frequency features. In practice, the width of the weight function should be tuned along with the radius of support, *h*. Both of these parameters affect the interpolation scheme. Generally,

- The wider the weight function, or the larger the value of *h*, the more smoothing results in field value approximations. This can be used to advantage to suppress numerical noise in the computations, but care must be taken not to suppress smaller scale, real features in the flow.

- The larger the value of *h*, the greater the number of nodes included in the least squares interpolation of field values for a given node under consideration, which results in smoother results at the cost of increased computation time.

- Conversely, the smaller the value of *h*, the fewer the number of nodes included in the least squares interpolation, which increases resolution in the sense that smaller scale flow features are preserved at the risk of increased numerical noise. Computation time is also reduced.

Tuning is an important process in any numerical simulation and these characteristics of the weight function should not be left out of the process in the DPM. Any useful implementation of the DPM should allow the user to easily adjust these parameters while tuning the simulation. This approach has been adopted in the implementation used throughout this research. The bottom line is that you must trade off numerical stability for speed and resolution. Keep in mind that DPM imposes no restrictions on the choices of weight function and support radius, *h*, for nodes at different locations within the same simulation. For example, in a flow simulation where the fluid domain is large relative to the actual area of interest, you can use a finer weight function and support radius in the area of interest while using larger support radii or wider weight functions further away from the area of interest. This approach facilitates resolution where you want it while striking a balance in terms of computation burden.

Numerical Implementation

Numerical implementation of the DPM generally involves marching the simulation through time solving the governing equations at each time step. For time marching I tested several explicit and implicit methods before finally settling on a fourth order Runge-Kutta scheme as a compromise in computational efficiency and stability. The major steps in numerical implementation are as follows:

1. Distribute computational nodes throughout the problem domain and initialize all field variables to initial and boundary conditions. Newman and Dirichlet boundary conditions can be handled by specifying the appropriate values at all boundary nodes. Rigid boundaries are treated as computational points with specified field values (pressure, density, velocity, or derivatives thereof) and are included in all interpolation and gradient computations. Momentum equations are not solved for at boundary nodes, although boundary nodes may have a prescribed velocity of something other than zero.

2. Start time marching scheme using appropriate time step sizes considering Courant condition and integration scheme.

3. At each time step fit shape functions for velocity, pressure, and density at each node given its nearest neighbors and store coefficients.

4. Used stored coefficients to compute gradients for each node and solve momentum equations.

5. Evolve density at each node using continuity equation and required gradients.

6. Compute new pressures at each node using state equation.

7. Repeat steps 3 through 6 for each time step.

There are some differences in numerical implementation depending on whether an Eulerian or Lagrangian approach is taken. In the former, momentum equations are solved first without convection terms, which are then solved for at half steps. The average convection results over a time step are then used when computing the new velocities at each node. Convection of other fluid properties are handled in a similar manner. In the latter case, convection terms are excluded altogether and upon computing new velocities, node positions are updated by an additional integration step. Here, other fluid properties are carried along with each node and they act as fluid particles very much like SPH particles.

Boundaries

In general rigid boundaries in DPM are treated by distributing computational nodes over boundary surfaces in much the same way computational nodes are distributed within a fluid domain to model the fluid. Once this is done, the nodes representing boundaries are treated very much like fluid nodes during numerical integration with some special treatments applied depending on the nature of the boundary conditions at those boundaries.

Rigid boundaries must always have a specified velocity for every node representing the boundary. The specified velocity need not be zero. In any case, the momentum equations used to update velocities every time step are not solved for the rigid boundary nodes. However, the pressure and density equations are used to compute pressure and evolve densities at rigid boundaries. In an Eulerian framework, nothing more need be done. In a Lagrangian framework, the rigid boundary nodes must be repositioned depending on the specified boundary velocity. This is achieved by integrating nodal velocities once to obtain new positions for the boundary nodes.

DPM does a better job at handling wall shear forces, i.e., no slip boundary conditions, than SPH. This aspect is discussed more fully in a moment (see upcoming section entitled *Viscosity*). Further, the no-flow-through condition is handled better in DPM. In SPH repulsive forces, Leonard-Jones forces, are modeled at rigid boundaries to keep fluid nodes from penetrating rigid boundaries. As stated earlier these repulsive forces sometimes exert very large forces on fluid particles leading to numerical instabilities and unrealistic flow patterns. DPM does not require such repulsive forces and the no-flow-through condition is handled naturally given specified boundary velocities and DPM's interpolation scheme.

Inflow and outflow boundaries are treated similarly to rigid boundaries where inflow and outflow nodes are used to represent the boundaries. Inflow nodes can be given prescribed inflow velocities or pressures. If inflow velocities are specified then the momentum equations are not used to solve for new velocities at inflow nodes; however, the pressure and density equations are solved at inflow nodes. If inflow pressures are specified then the momentum equations are used to compute velocities at inflow boundaries, but the pressure and density equations are not solved at inflow boundaries.

Outflow boundaries are treated similarly. Moreover, outflow gradients may be specified in which case velocities, pressure, and densities are solved for at outflow nodes where account is taken of the specified gradients at the outflow nodes. This is the only type of boundary condition that requires some special treatment in the way gradients are computed. Essentially, gradients at such nodes are not computed, since they are specified, and these specified gradients are used in the evolution of velocity, pressure, and density.

So far this research has not implemented free surface boundaries in DPM. However, such boundaries may be simulated in DPM in a manner similar to how they are treated in SPH. Differences would be required depending on whether or not an Eulerian or Lagrangian framework is used. That said, multi-fluid interfaces have been modeled in this research as will be discussed later in Chapter 3.

Viscosity

As discussed earlier in Chapter 1, most implementations of SPH use an artificial viscosity model to introduce shear and/or bulk viscosity into the momentum equations. Some researchers have incorporated so-called *real* viscosity into SPH formulations as discussed in Chapter 1, but those models come with added complexity and computational expense.

Part of the challenge implementing real viscosity in SPH is due to the requirement to transform viscous forces into equivalent pressure forces. Since DPM computes second derivatives using its interpolation scheme and applies such derivatives directly to the Navier-Stokes equations no transformation of forces is required; the Navier-Stokes equations are used directly. This makes handling viscous forces very easy and applied to both fluid nodes and rigid boundary nodes without exception since each node carries with it a coefficient of viscosity, it's own velocity and velocity gradients relative to its surrounding nodes.

Nearest Neighbor Search

A critical aspect of implementing DPM (as well as SPH) is in determining the nearest neighbors for every node in the simulation. Nearest neighbors are those nodes, the k^{th} nodes, that are in the vicinity of any given node under consideration, the i^{th} node, as illustrated earlier in Figure 2.3.

To be more precise, nearest neighbors are those nodes that fall within the *i*th node's radius of influence, that is, those are the nodes that somehow have influence of node *i* and are thus included in all interpolation and gradient calculations for node *i*. This radius of influence is controlled by the parameter *h*, also called the *smoothing length*, as discussed earlier. If a *k*th node is not within the *i*th node's smoothing length, then we don't want to keep track of it as a neighbor to *i*. Remember, for each time step, and for every node, we have to iterate through all nearest neighbors of every node in a simulation. In a naïve implementation on could iterated through every node in a simulation for every *i*th node while computing gradients; however, this is incredibly slow. I've already discussed in Chapter 1 how such an approach is order $O(N^2)$.

A better approach is to somehow keep track of every node's nearest neighbors and only iterated through that list of neighbors when interpolating and computing gradients. In the DPM I've implemented a background zone approach for keeping track of neighbors. The application of this zone approach is somewhat different depending on the framework used, i.e., Eulerian versus Lagrangian.

At the start of a simulation a background grid of specified resolution is superimposed over the fluid domain being simulated as illustrated in Figure 2.8.

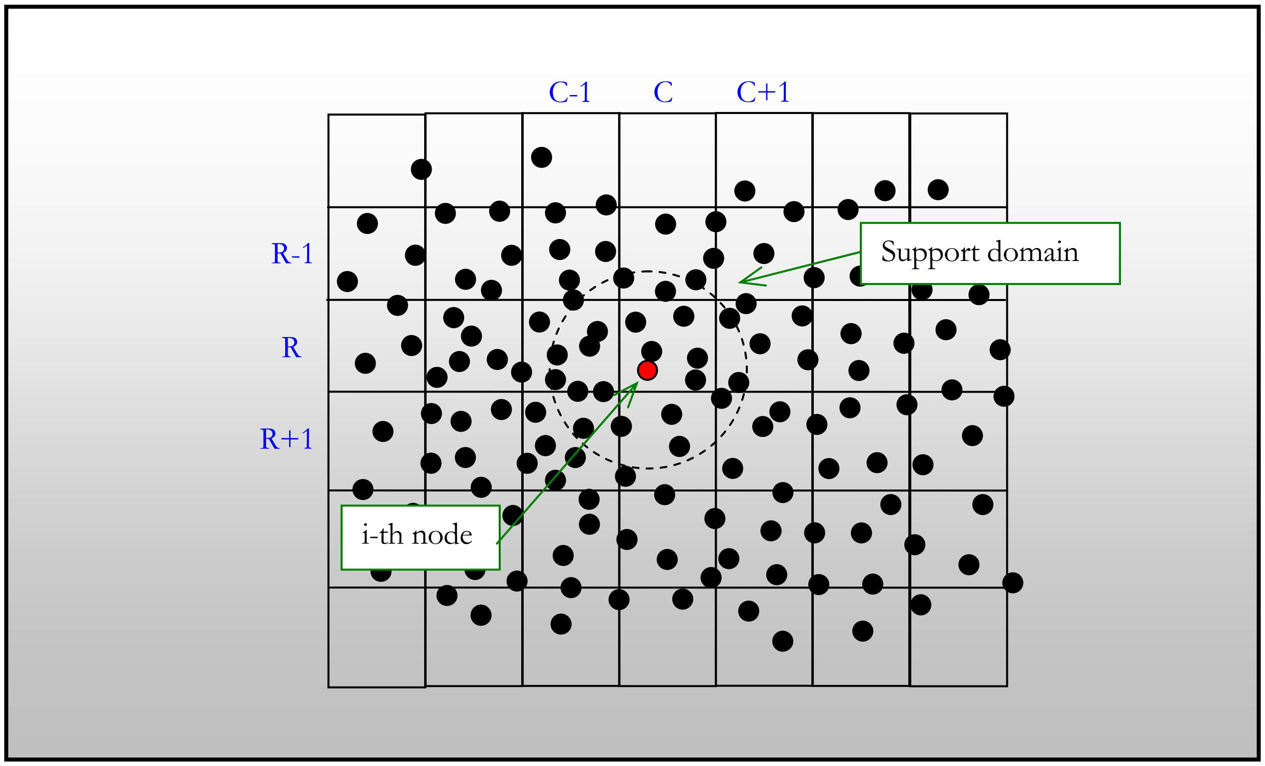


Figure 2.8: Background grid.

The grid is comprised of cells and the resolution is such that each cell is at least *h* tall by *h* wide. In practice, it is helpful to make each cell $h+\delta$ by $h+\delta$ to avoid difficulties when a node falls directly on a grid line. After this grid is constructed one iteration through each node is performed to determine what cell, specified by (R,C), each node falls within. This cell value is stored for each node.

To determine the neighbors for each node, the i^{th} node for example referring to Figure 2.8, one need only consider the k^{th} nodes that fall within plus or minus one row and column from node *i's* stored row and column. If Figure 2.8, node *i* falls in cell *(R,C)* and all of its neighbors fall within cells *(R-1, C-1)* to *(R+1, C+1)*. Once the neighbors are determined, pointers to those neighbors are stored in a linked list. Each node possess a linked list of pointers to its neighbors and during a simulation only those neighbors are considered when computing gradients for each i^{th} node.

In an Eulerian framework, where nodes do not move, this nearest neighbor search need only be performed once at the start of the simulation. However, in a Lagrangian framework, where nodes actually move neighbors may change so this nearest neighbor search must be performed every time step. This adds one additional iteration through all nodes in the simulation each time step, but the payoff is substantial in that you avoid iterating through the entire node list once for every node in the simulation per time step.

Time Stepping

As discussed in Chapter 1, Monaghan has shown, through analysis of the Navier-Stokes equations that the proper choice of *c* used in the state equation in SPH can be obtained from the following:

$$c^2 = \max\left[\left(\frac{V_o{}^2}{\delta}\right), \left(\frac{\upsilon V_o}{L_o\delta}\right), \left(\frac{FL_o}{\delta}\right)\right]$$

where

$$\delta = \frac{\Delta\rho}{\rho_o}$$

Here, V_o is the velocity scale, L_o is the length scale, *F* is the body force per unit mass acting on the fluid, and *v* is the kinematic viscosity. In the DPM, the same conditions apply and one generally tries to keep the artificial Mach number around 0.1 for incompressible flows. The selection of the sound speed, *c*, has an influence on the step size than can be taken during a simulation. For a given minimum node spacing, δs, the maximum step size should not exceed, $\delta s/c$. This basically restricts a wave from traveling a distance equal to or greater than the minimum node spacing over a single time step.

Another useful limit I've found helpful for retaining numerical stability in DPM is to limit step sizes to less than $\delta s/|V|$, where $|V|$ is maximum speed of any given node in the simulation. In this research, I've implemented an adaptive step size algorithm adopted from [72] to help speed up simulations where larger time steps may be taken when there's lower risk of numerical instability.

For every time step, I take three steps: one step is taken from *to* using a time step δt; then again from t_o two steps are taken using $\delta t/2$. Results for velocity, pressure, and density are saved after each step from *to* to *to*+δt. These results are compared and their differences are computed. These differences represent the errors due to taking one large step versus two smaller steps. If this error is within a prescribed tolerance then the step size is increased for the next simulation step. The step size is increased according to the formula,

$$\delta t_{new} = \delta t_{previous}\left(\frac{tolerance}{error}\right)^{0.2}$$

If it is greater than the prescribed tolerance then the step size is reduced and the simulation step retaken. The step size is reduced according to,

$$\delta t_{new} = \delta t_{previous}\left(\frac{tolerance}{error}\right)^{0.2}(b)$$

where *b* is usually taken at 0.97.

Chapter 3

INITIAL TESTS OF THE DISTRIBUTED POINTS METHOD

During development of the distributed points method (DPM) I implement several benchmark tests to help ensure the method was performing correctly. While each test yielded numerical results, I was primarily interested in qualitative assessment of certain aspects of the method, namely, the method's ability to: handle viscosity and rigid boundaries; to model inflow and outflow boundary conditions; to capture circular flow patterns; to perform in Lagrangian or Eulerian frameworks; to perform with ordered or randomly distributed nodes; to handle variable smoothing lengths; to model multiple fluids in the same simulation capturing effects such as buoyancy; and, to model wake development behind a moving object (not necessarily a solid object).

Moreover, these tests were used to debug the code implementation of DPM and to tune certain aspects of the implementation such as the nearest neighbor search algorithm and the numerical integration scheme. In this respect the benchmark tests were used as development tools. The code was developed and refined through iterative execution of these benchmark tests. While several benchmark problems were modeled, three in particular -- Poiseuille flow between flat plates, lid driven cavity, and a multifluid test – proved very useful in testing the above cited capabilities of DPM. These tests were constructed so as to be easily implemented and easy to check relative to theory, previously published numerical or experimental results, and intuition.

Poiseuille Flow

The first problem modeled was that of the two dimensional flow between two infinite flat plates. For low Reynold's numbers, *Re*, this problem has a straightforward solution for the velocity profile which can be derived from the Navier-Stokes equations. Many standard fluid mechanics texts present such a derivation so it will not be repeated here. Figure 3.1 illustrates the problem.

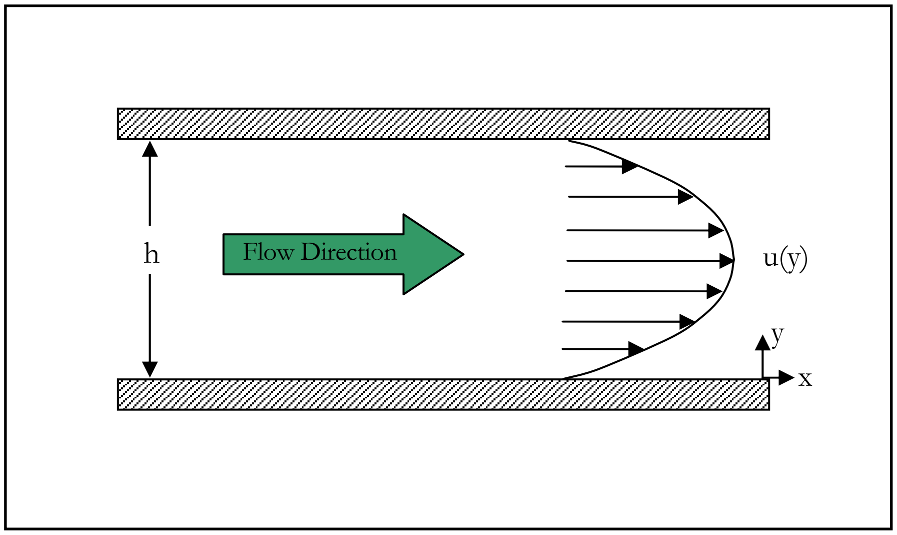


Figure 3.1: Poiseuille flow between two flat plates

For laminar flow, the velocity profile, *u(y)*, is described by a parabola of the form,

$$u(y) = -\frac{1}{2\mu}\frac{dp}{dx}\left(hy - y^2\right)$$

where, μ is the fluid viscosity, *dp/dx*, is the pressure gradient in the x-direction, and *h* is the distance separating the two plates. For the purpose of these tests, it is convenient to rewrite this result in the following form,

$$u(y) = \frac{\rho g_x}{2\mu}\left(hy - y^2\right)$$

where, we have replaced the pressure force per unit volume in the x-direction term, *-dp/dx*, with an equivalent force per unit volume, ρg_x, where g_x is a body acceleration in the x-direction.

Figure 3.2 illustrates how this problem was set up in a DPM implementation. The dots represent fluid nodes distributed evenly throughout the fluid domain. The upper and lower rectangular blue areas correspond to rows of boundary nodes also evenly distributed in the horizontal direction. The dots with white borders represent virtual inflow and outflow nodes to model periodic boundary conditions.

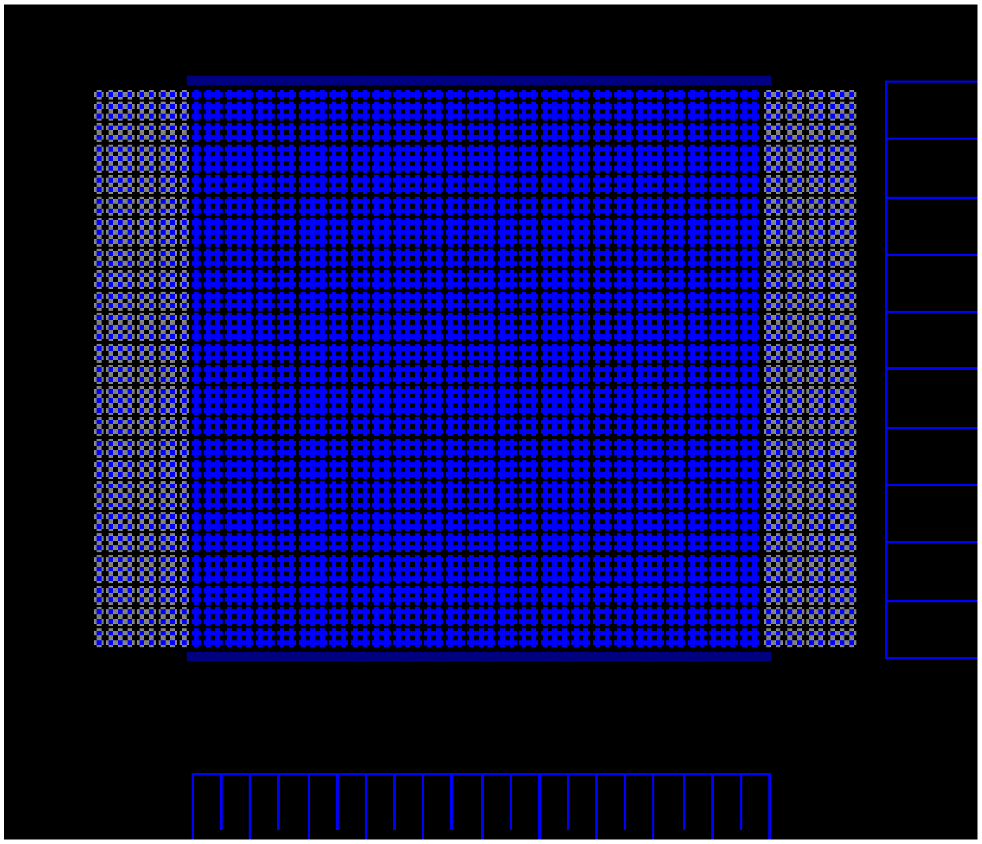


Figure 3.2: Lagrangian model

This set up is for a Lagrangian formulation where the nodes will move throughout the simulation. As nodes exit the outflow boundary on the right, they will be reintroduced into the inflow boundary on the left. Nodes representing the upper and lower rigid plates will remain stationary under a specified zero velocity boundary condition. The particulars of this model are summarized in Table 3.1.

Table 3.1: Poiseuille flow parameters

ρ	999.83 kg/m^3
μ	9.9983x10^{-4} kg-s/m^2
g_x	8.0x10^{-5} m/s^2
h	0.01 m
Re (based on h)	10.0

Figures 4.3 and 4.4 show DPM results for both Lagrangian and Eulerian formulations. The color scales in both range from blue to red with blue corresponding to zero m/s and red corresponding to 0.001 m/s.

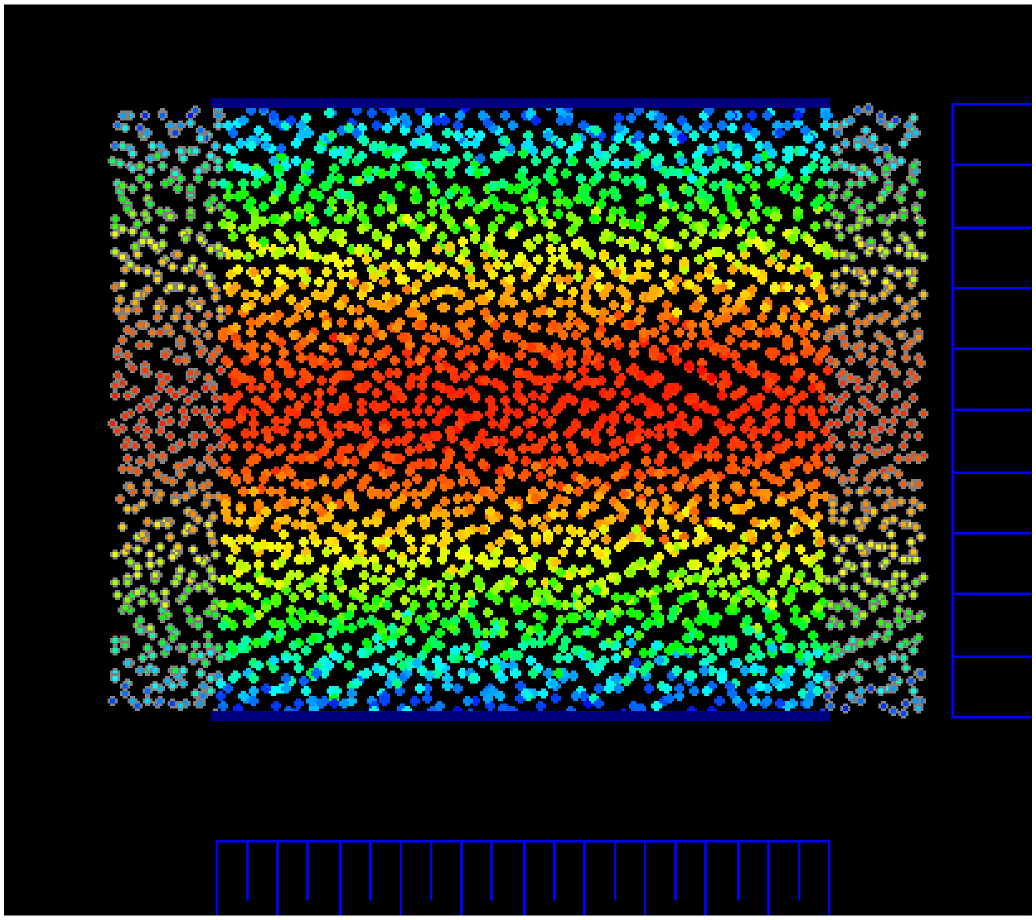


Figure 3.3: Lagrangian model, Re = 10.0

The Lagrangian model shown in Figure 3.3 includes 3245 nodes. The simulation was started with all nodes at rest under the influence of the specified body acceleration in the x-direction. The snapshot shown in Figure 3.3 was taken after the flow had reached steady state. The apparent disorder in the nodes relative to the initial setup shown in Figure 3.2 is a characteristic of the Lagrangian approach where nodes are advected through the fluid subject to the resulting velocity field at each time step. The color bands reveal that the maximum velocity occurs at the midplane between the two rigid boundaries as expected. Further, nodes directly adjacent to the rigid boundary remain virtually still due to the no slip boundary condition modeled through viscous action between fluid and boundary nodes as discussed in the previous chapter.

Figure 3.4 shows a snapshot of an Eulerian model of the same problem. This particular model uses 1052 nodes. Given the same problem parameters, the results of the Eulerian model are essentially identical to the Lagrangian model. The snapshot in Figure 3.4 was taken after the simulation reached a stead state.

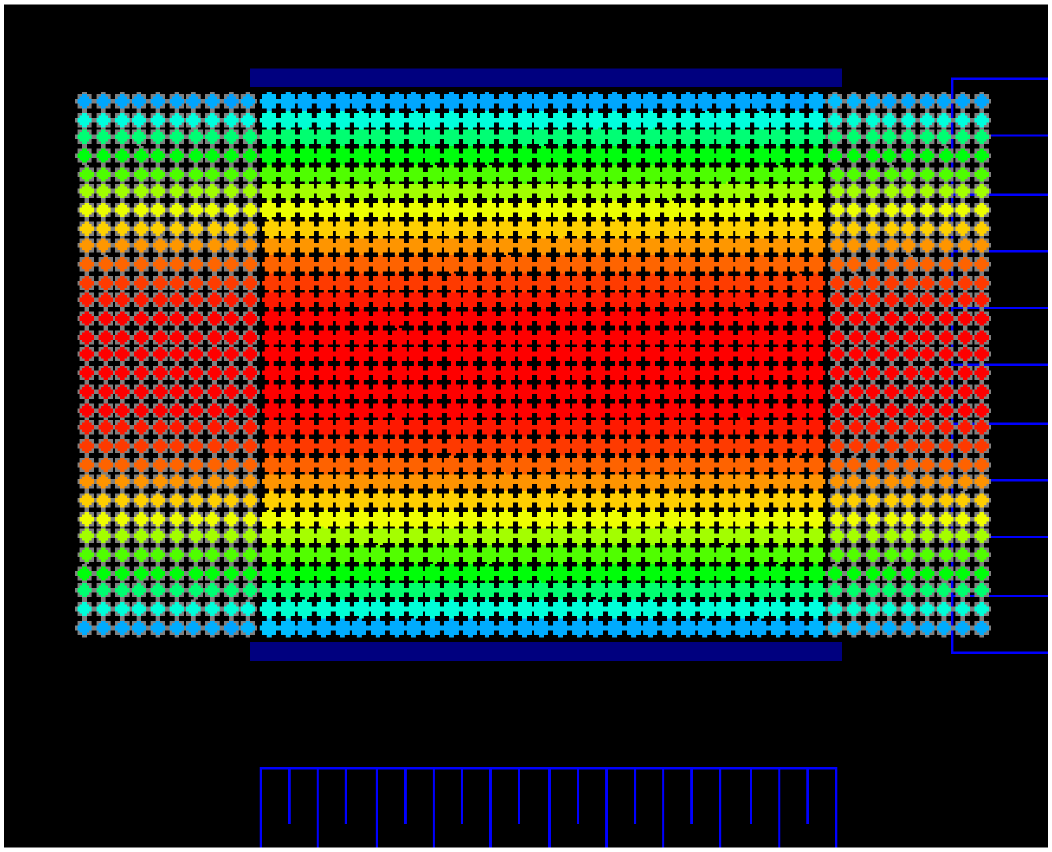


Figure 3.4: Eulerian model results, Re = 10

In this Eulerian model all nodes remain stationary but the convective terms in the Navier-Stokes equations are included in the formulation in order to advect the velocity field. Here again, we see that the maximum velocity occurs at the midplane between rigid plates and goes to zero as we near the plates. The Eulerian formulation for this particular problem turns out to be easier to code owing to the simplified way in which inflow and outflow boundaries are handled: there's no need to keep track of nodes exiting and re-entering the flow domain in the Eulerian approach.

Figure 3.5 shows some numerical results compared to theoretical results. In general, this comparison shows excellent agreement giving confidence that the approach taken in DPM to model viscosity is performing as it should.

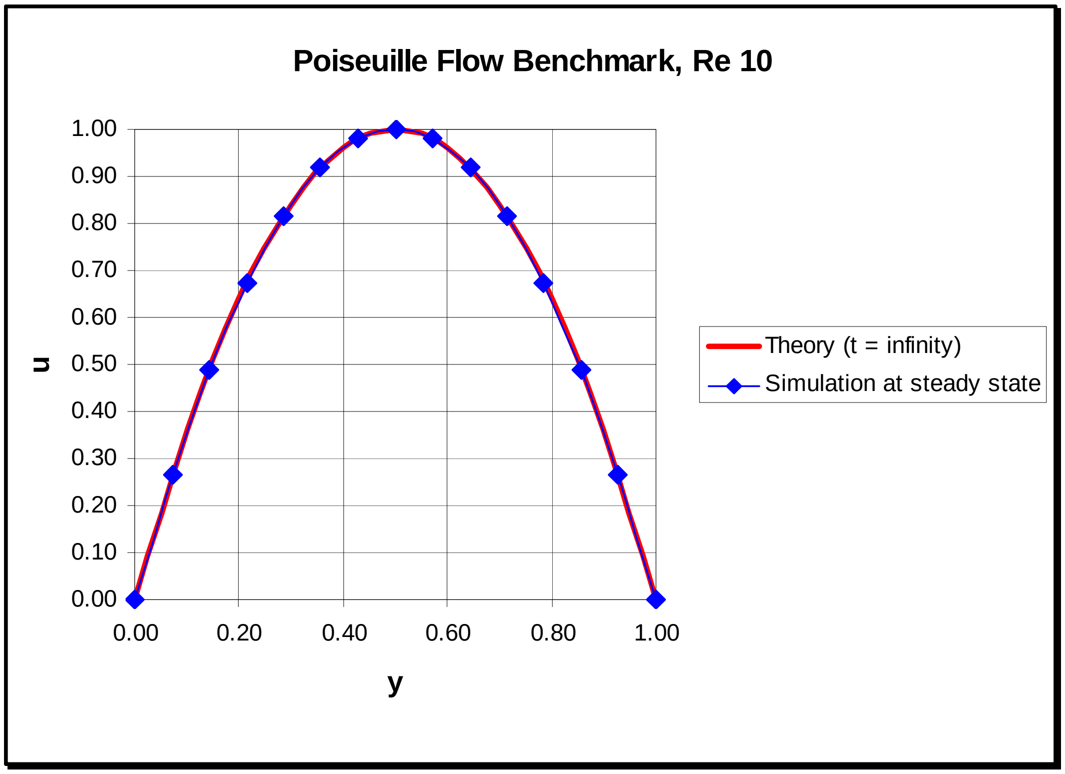


Figure 3.5: Poiseuille flow results comparison

There are several other areas of performance that were validated by this test. First, this test showed that DPM can work in both Lagrangian and Eulerian formulations. While there are some programming differences in the way inflow and outflow boundary conditions are handled, it was relatively easy to switch from one formulation to the other. The code developed in this research includes a Boolean flag that can be toggled to switch between approaches. In the Lagrangian approach convective terms in the Navier-Stokes equations are excluded but an additional integration of the velocity field is performed to advect the fluid nodes. In the Eulerian approach this additional integration is not required, however, the convective terms are included in the Navier-Stokes equations.

Secondly, these tests showed that viscosity can be modeled using the DPM approach. The results illustrated the development of viscous boundary layers along the upper and lower plates until they converged to form the steady state velocity profile predicted by theory. Numerical results are in excellent agreement with theory.

Third, rigid boundaries are, so far, adequately modeled by distributing nodes along those boundaries. No slip boundary conditions are easily specified by setting all boundary node velocities to zero and excluding those nodes from integration of the momentum equations. Further, given that SPH-style Leonard Jones repulsive forces are not required in DPM's Lagrangian formulation, fluid node velocities do not exhibit the bumpy drag force seen in some SPH implementations. [29]

Lid Driven Cavity

This next test involves the problem of lid driven cavity flow. A rectangular cavity is filled with fluid and contained under a lid as illustrated in Figure 3.6. The lid of the container moves to the right at some speed, *U*. Due to viscous action between the lid and the fluid, a shear layer will develop inducing a circular flow pattern within the cavity. This particular problem is well documented in the technical literature where many numerical and physical test results are available for study.

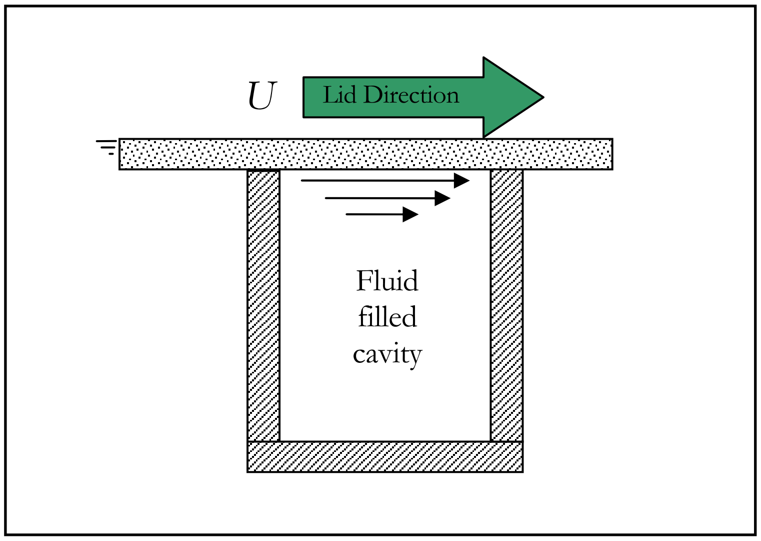


Figure 3.6: Lid driven cavity illustration

As the lid moves to the right, shear forces begin to pull the fluid adjacent to the lid towards the right under viscous action. A shear boundary layer develops and as the flow continues to the right, it must turn downward due to the right boundary. Since the fluid is contained on all sides, a circulation sets up within the cavity. The center of main circulation is biased towards the top of the cavity consistent with SPH simulations of the same problem.[74] High resolution numerical tests reveal that two regions of counter circulation form in the lower left and right corners. [75]

This problem was selected primarily because it represents a good test of the DPM's ability to model viscous flow. The flow in the problem is entirely driven by viscous forces. Moreover, this particular problem usually presents some difficulties for mesh free methods such as SPH where particles tend to evacuate the upper left corner of the fluid domain leaving that area of the domain undefined in Lagrangian numerical simulations. Eulerian approaches are better suited to handling this problem. There are other reasons for selecting this problem as will be discussed shortly.

The DPM model for this problem is relatively simple. Figure 3.7 illustrates the initial node set up where the fluid nodes are represented by round blue circles. Rigid boundary nodes are represented by solid squares. The blue squares are stationary boundaries, whereas the red squares represent the moving lid.

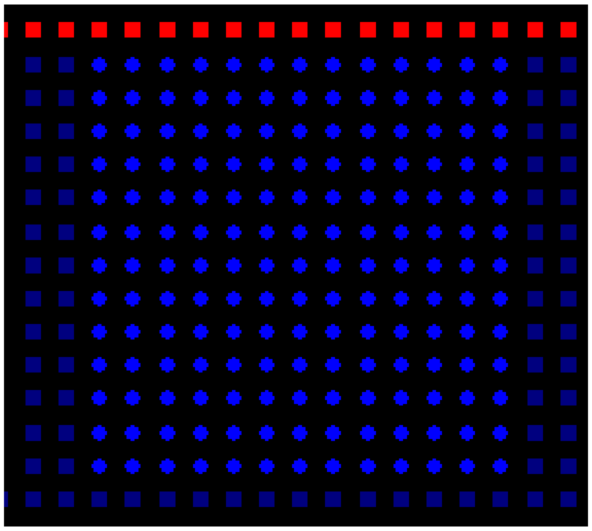


Figure 3.7: Lid driven cavity set up

This is a fairly low resolution model comprised of 252 nodes. Initial simulations were conducted using a Lagrangian formulation. Figure 3.8 illustrates typical results, in this case, for Re = 10.

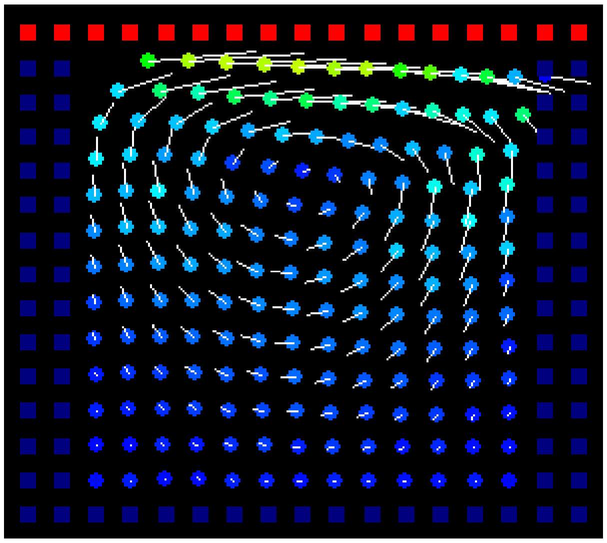


Figure 3.8: Lagrangian test results

This simulation does a good job of capturing the dynamics of the flow whereby viscous shear at the top of the cavity sets up a circular flow pattern within the cavity. The center of circulation is also where expected and biased towards the top of the cavity. This test also reveals a troublesome aspect of a Lagrangian approach. Like SPH simulations of this same problem, the DPM results show how Lagrangian nodes, or particles in SPH, tend to evacuate the upper left corner of the cavity leaving the area numerically unresolved.

One approach to remedy this situation is to periodically redistribute the nodes so as to fill the voids. However, this requires interpolation of all field values – pressure, density, velocity – at each new node location. This, extra, step invariably smoothes the results each time the redistribution is applied. The end result is artificial diffusion.

This sort of problem is better suited to an Eulerian approach where resolution in the entire flow field is guaranteed. Unlike SPH, DPM can readily handle the lid driven cavity using an Eulerian formulation. Figure 3.9 shows some typical results from a higher resolution (3478 nodes) Eulerian model of the lid driven cavity problem.

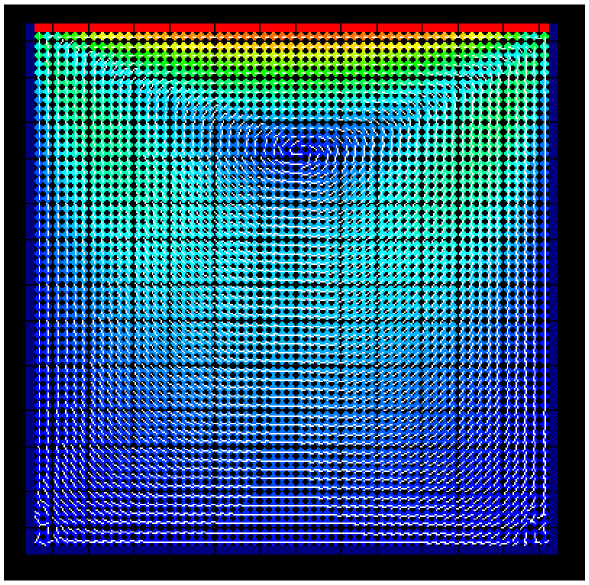


Figure 3.9: Lid driven cavity Eulerian model velocity plot

Figure 3.9 shows the velocity field after the simulation reached a steady state. The colors represent speed where blue is zero speed and red is the specified speed of the lid. The white tick marks indicate the direction of the velocity vector at each node. The larger circulation area is clearly apparent towards the upper part of the container. Moreover, this model captures two areas of counter circulation in the lower corners of the cavity. Higher resolution test in those areas would be required to fully capture the flow there; however, the present results are positive in that those counter circulation areas have been captured.

Figure 3.10 illustrates how this DPM simulation captures other flow field values. In this figure, the pressure field corresponding the above velocity field is presented.

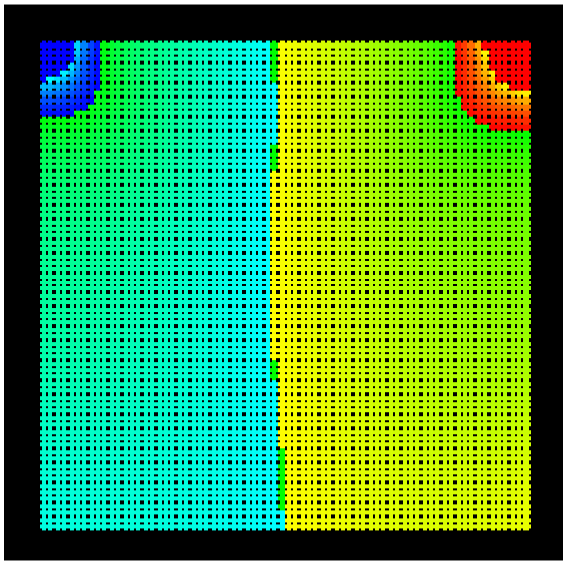


Figure 3.10: Lid driven cavity Eulerian model pressure plot

Intuitively, this is an expected pressure distribution. The blue in the upper left corner represents a region of low pressure, which corresponds to the region where fluid is being tugged to the right and drawn up from below. The red region in the upper right corresponds to a region where the fluid having been accelerated and pushed to the right, hits the right side of the cavity and is deflected downward.

Multi-fluid Test

This multi-fluid test consists of a square, closed container containing two fluids of different densities. At the start of the simulation, a square slug of the denser fluid is located at the center of the container surrounded by the less dense fluid. Figure 3-11 and 3-12, illustrate the container with initial slug of dense fluid at its center. Once the simulation is started, the denser fluid is allowed to sink and settle under the influence of a gravity. The purpose of this test is to see how well the DPM method deals with two fluids of differing density in the same model. Handling this simple case is a first step towards adapting the DPM method for cavitating flow problems.

In Figure 3-11, the red dots represent the denser fluid, while the blue dots represent the less dense fluid. The blue rectangles surrounding the blue dots represent the container's, rigid boundaries.

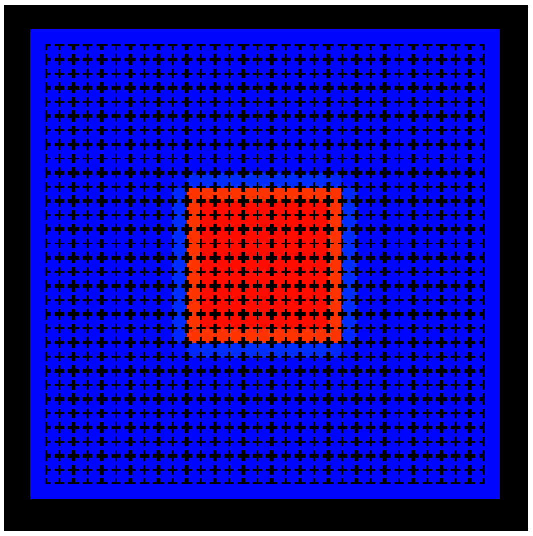


Figure 3-11: Mulit-fluid test initial set up

Figure 3-12 illustrates Lagrangian tracker particles that are used to track the denser fluid. I used an Eulerian formulation for this problem with Lagrangian tracker particles for flow visualization only.

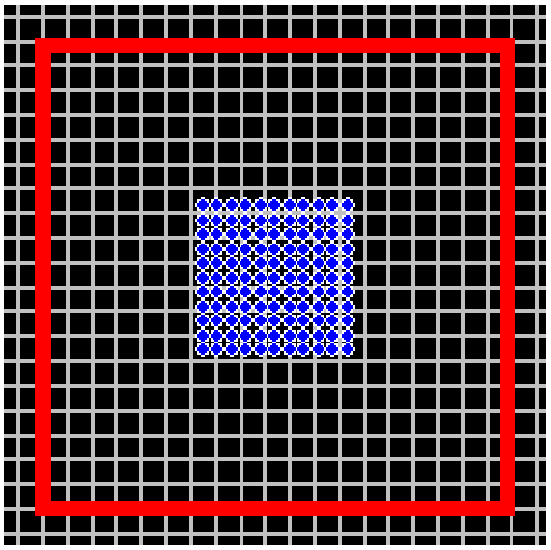


Figure 3-12: Multi-fluid test initial set up with tracker particles

The specific parameters for this problem are shown in Table 3.2. The density ratio in this case is 2:1. While not a large difference in density, it serves the previously stated purpose of this exercise. Chapters 5 and 6 show how DPM was adapted to handle density ratios of 1000:1.

Table 3.2: Multi-fluid flow parameters

ρ dense fluid	999.83 kg/m^3
ρ light fluid	499.92 kg/m^3
μ both fluids	9.9983x10^{-4} kg-s/m^2
g_y	9.8 m/s^2

Both fluids in this test are assumed to have the same viscosity coefficient. However, that assumption isn't necessary as will be discussed later in Chapter 5. The two fluids do have different densities. In order to handle this difference nothing was done differently in the DPM formulation from the original single fluid method discussed in the previous chapter with the exception that the nodes were initialized with differing densities – the nodes representing the initial location of the dense fluid were given an initial density of 999.83 kg/m^3, whereas, all other nodes representing the lighter fluid were given a density of 499.92 kg/m^3. The DPM formulation naturally handles density differences. All nodes are included in the interpolation calculations for all field values as though all nodes were a single fluid. Differences in density will manifest themselves in the interpolation functions from which gradients will be computed.

Success here depends on the use of the weighted least squares approximation. Consider for example, the univariate example shown in Figure 3-12. The squares represent nodal densities plotted against distance relative to the ith node, which is the node for which we wish to compute the density gradient. The plot indicates a 2:1 difference in densities going from left to right. Assuming the support domain consists of all nodes shown, an un-weighted, linear fit results in the dashed line shown in the figure.

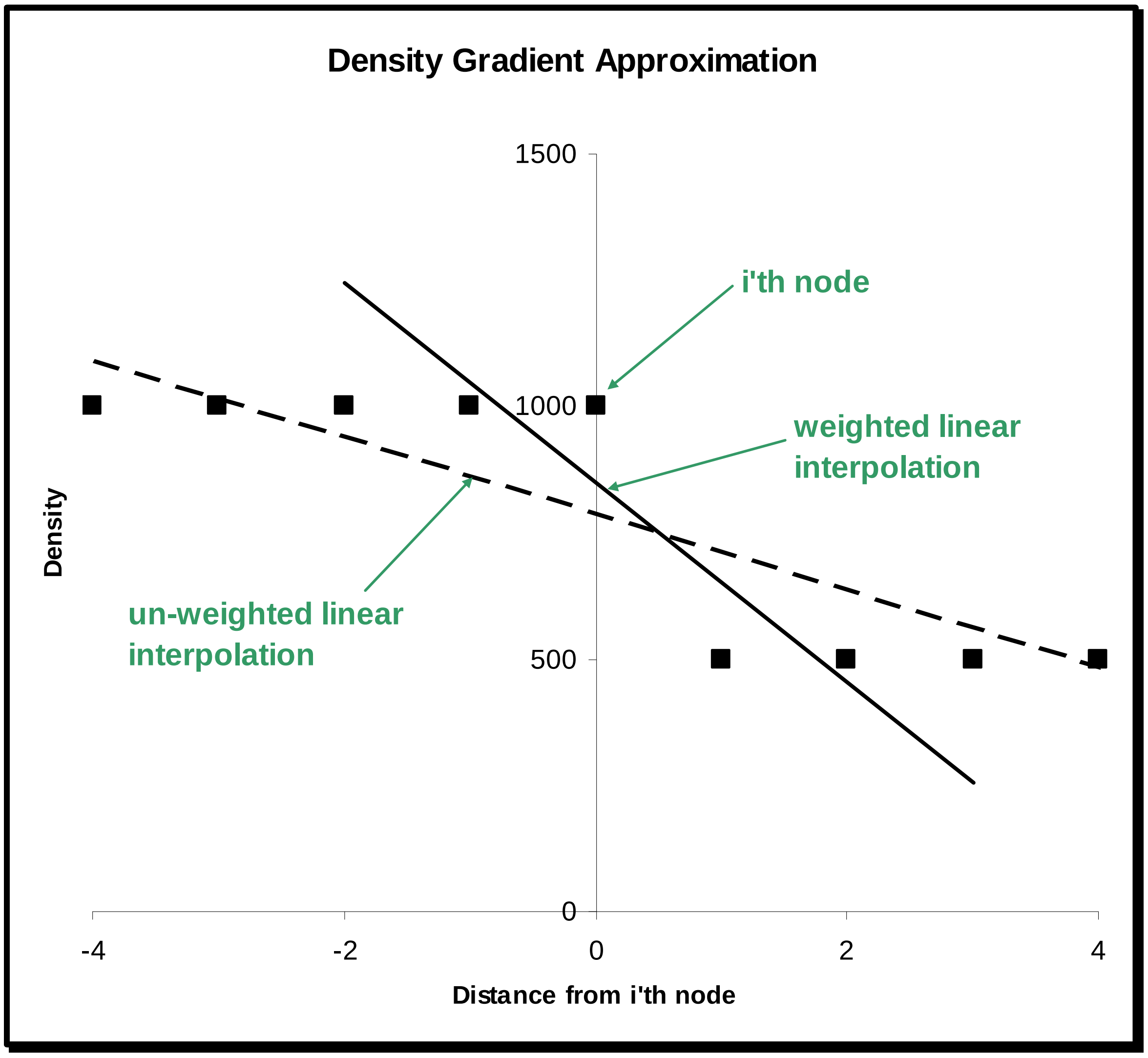


Figure 3-12: Density interpolation example

If this un-weighted fit line is used to compute the density gradient at the ith node, it's clear the resulting estimate will be greatly in error. However, a weighted linear fit using a linear weight function results in the solid line shown in the figure. Using this fit line will result in a much better gradient approximation. A Gaussian weight function giving more weight to the ith node would result in an even better approximation. Remember, the purpose of the least squares fit is not to approximate the field value itself, density in this case, but to approximate the gradient of the field value. It is clear that selection of the weight function is an important consideration for multi-fluid modeling using DPM.

Turning back towards the multi-fluid test, Figure 3-13 shows simulation results 0.5 seconds after the slug of dense fluid has begun its descent towards the bottom of the container.

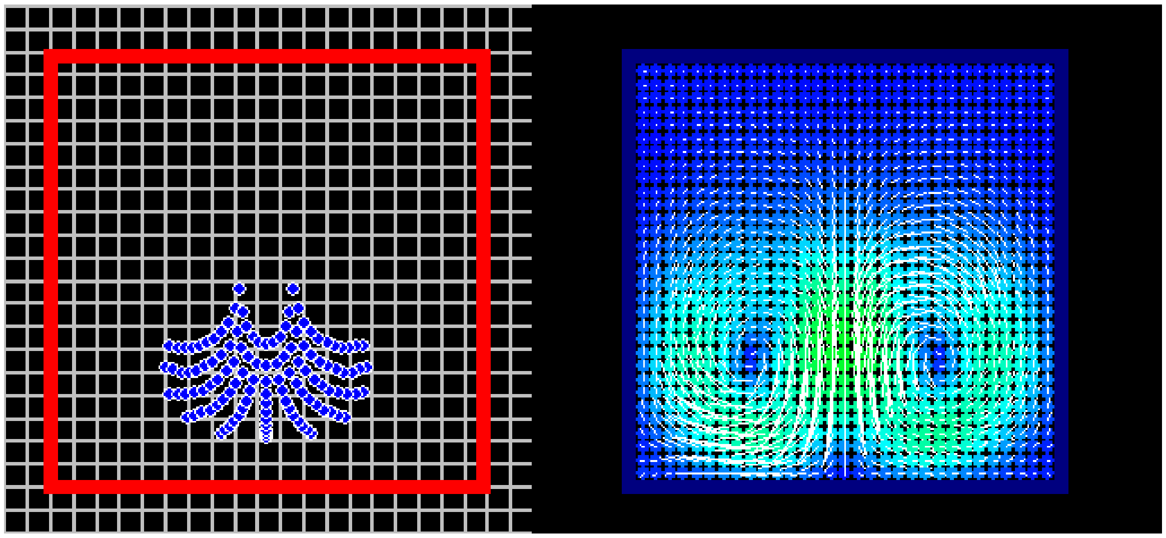


Figure 3-13: Multi-fluid results

The left of Figure 3-13 shows the Lagrangian tracker particles. The particles appear to take on a droplet formation as expected. The right side of Figure 3-13 shows the velocity field in both fluids at the same time instant. This result shows how symmetric circulation exists to either side of the dense droplet. As the droplet descends the less dense fluid below is pushed down and deflected sideways at the bottom of the container. The less dense fluid then rises and curls around the drop into its wake. This flow pattern is expected since this problem is essentially the reverse of a rising bubble problem where the flow pattern is very similar (albeit with the bubble rising instead of falling).

Figure 3-14 shows a sequence of density field plots showing the formation of the dense droplet. The images from left to right represent the droplet at 0.25 seconds and 0.5 seconds, respectively, after the start of the simulation. Blue dots represent the lighter fluid and red dots represent the denser fluid. It is clear from these density plots that there's some numerical smoothing of density across the interface from denser fluid to light fluid. The green dots represent densities midway between the dense and light fluid densities.

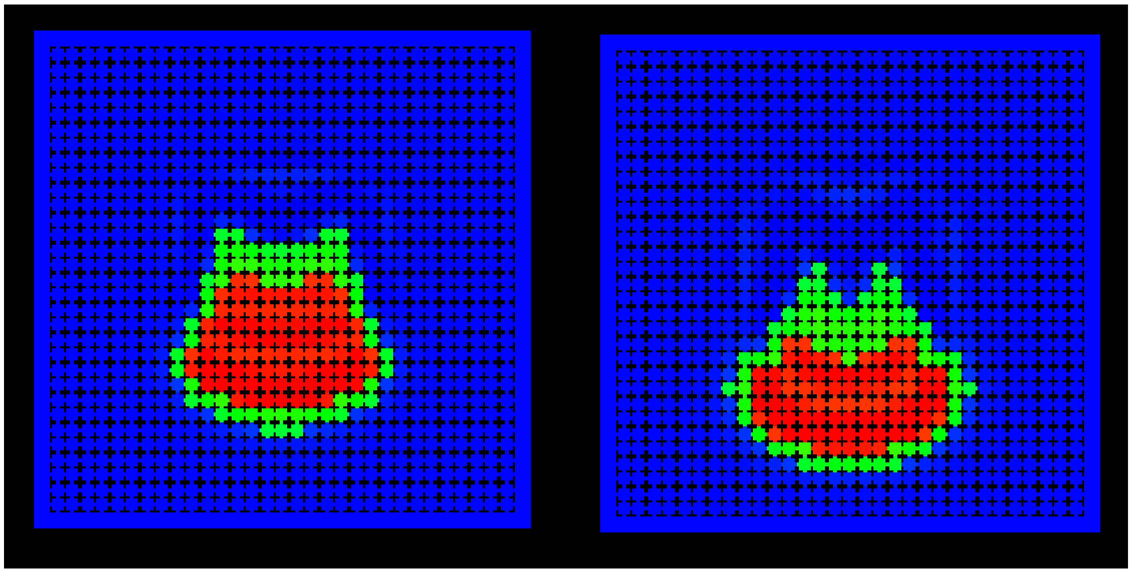


Figure 3-14: Droplet density plots

The extent of this smoothing is on the order of a smoothing length, *h*, and is more prominent in the wake of the droplet where the lighter fluid curls back on the droplet as discussed earlier. The smoothing along the density interface represents an averaging of the fluid densities to either side of the density interface.

This smoothing raises the question of mass conservation. While this simulation ran, total mass of the system was tracked. Given this is a closed system, the mass should remain constant and indeed from the start to the end of the simulation, the total mass changed by only -0.65%.

One would expect the dense fluid to settle on the bottom of the container if the simulation were allowed to reach a steady state. Indeed this is the case as illustrated in Figure 4-15.

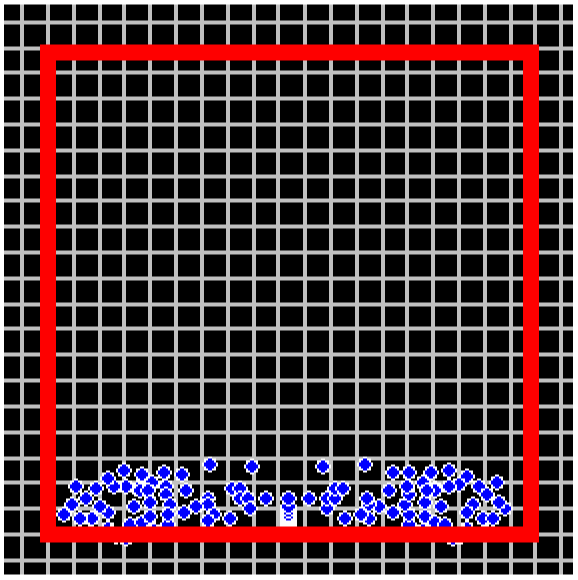


Figure 4-15: Settled fluid

This problem has shown that DPM is capable of handling multiple fluids without modification to its basic formulation. The only extra consideration required for this test was a careful selection of weight function to resolve the sharp density gradient at the interface of the two fluids. The thickness of the smoothed density interface is on the order of one smoothing length, which suggests that the thickness can be controlled by careful weight function selection or higher resolution using smaller smoothing lengths.

Chapter 4

2D CIRCULAR CYLINDER BENCHMARK

The tests discussed in the previous chapter, with the exception of the Poiseuille flow problem, are decidedly qualitative. In order to better measure DPM's performance and assess the method's sensitivity to certain elements I ran a systematic series of tests using two dimensional flow past a circular cylinder as a benchmark problem. These tests aim to model the unsteady flow around a two dimensional circular cylinder for Reynolds numbers up to 1000. The circular cylinder problem was selected because it is a well understood problem with many resources containing experimental information.

The geometry for these tests consists of a 2D circular cylinder oriented with its long axis perpendicular to the oncoming flow. The fluid domain is bounded by upstream, downstream, and side boundaries where free stream boundary conditions are applied (these are not solid wall boundaries). Figure 4.1 illustrates the simulation geometry where the cylinder is located toward the left hand side and the flow direction is from left to right.

Inflow boundary nodes were set to a constant inflow velocity. The upper and lower free stream boundary nodes were also set to the free stream (inflow) velocity. Zero gradient conditions were set at the outflow nodes and the field values of those nodes were allowed to evolve in accordance with the governing equations.

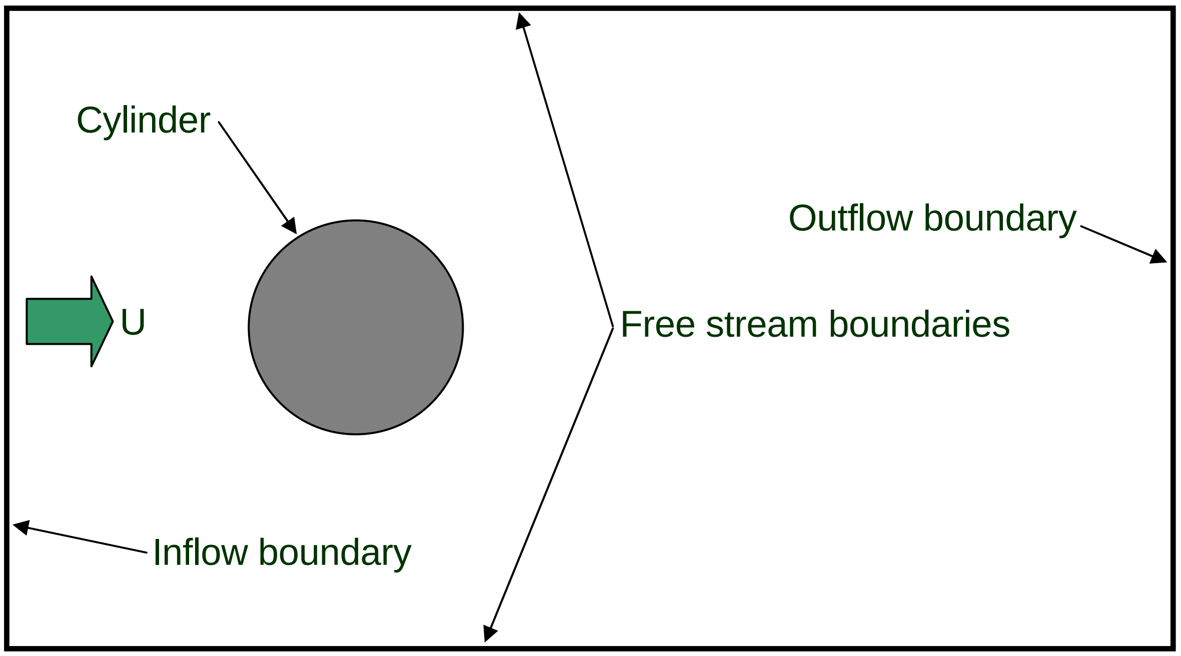


Figure 4.1: Circular cylinder problem geometry

Baseline Test

At the outset of this series of tests, a baseline case was established to provide a basis of comparison for all sensitivity tests. For the baseline case the downstream boundary was located 10 cylinder diameters aft of the cylinder. The upstream boundary was located 5 diameters upstream of the cylinder. The side boundaries were 5 diameters to either side of the cylinder. Baseline simulations were run at a Reynolds number of 125. This Reynolds number was chosen since, based on experimental results, it falls within a range where the vortex instability exists and vortex street inception is expected to occur. Baseline tests were run with an artificial sound speed chosen so as to yield an artificial Mach number of 0.1, which is within the range where compressibility effects are considered negligible. The baseline state equation was as follows:

$$P = c^2 \frac{\rho}{\gamma}\left(\frac{\rho}{\rho_o}\right)^{\gamma}$$

This is a typical barotropic state equation used in a variety of artificial compressibility methods. In this equation *c* represents the artificial sound speed, ρ represents the fluid density, ρ_o is a reference density, and γ is the specific heat ratio.

The baseline tests also included a linear weighting function. Resolution of the baseline tests was fixed at 4,579 nodes. An Eulerian approach was taken as discussed earlier.

The baseline test results were as expected. Through an initial ramp up period where the fluid velocity was steadily increased a fixed pair of stable vortices developed just behind the cylinder. As flow velocity increased further, those vortices stretched downstream while still maintaining their symmetric character. Once the fluid speed was fully ramped up to a corresponding Reynolds number of 125 these symmetric vortices started to display signs of instability. Subsequently, the instability turned into the fully developed vortex street as shown Figure 4.2. Qualitatively, the baseline test results agree with published experimental results. The observed development of symmetric vortices and then transition to the vortex street is precisely what was expected.

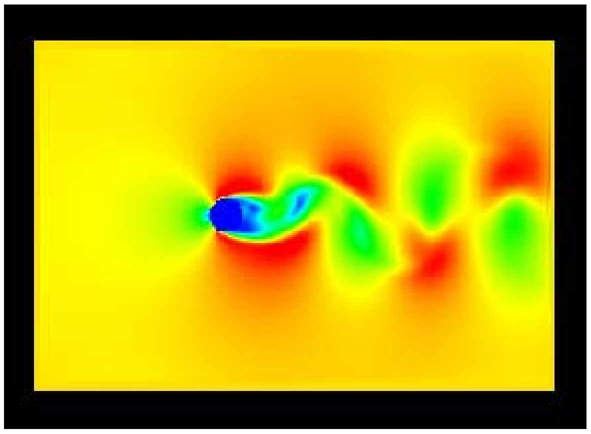


Figure 4.2: Baseline test velocity field

The colors in Figure 4.2 represent velocity intensity with warmer colors indicating higher intensity. Figure 4.3 shows a close up of the cylinder, where the little white lines represent velocity direction and magnitude. Trailing, asymmetric vortices are clearly visible in this Figure.

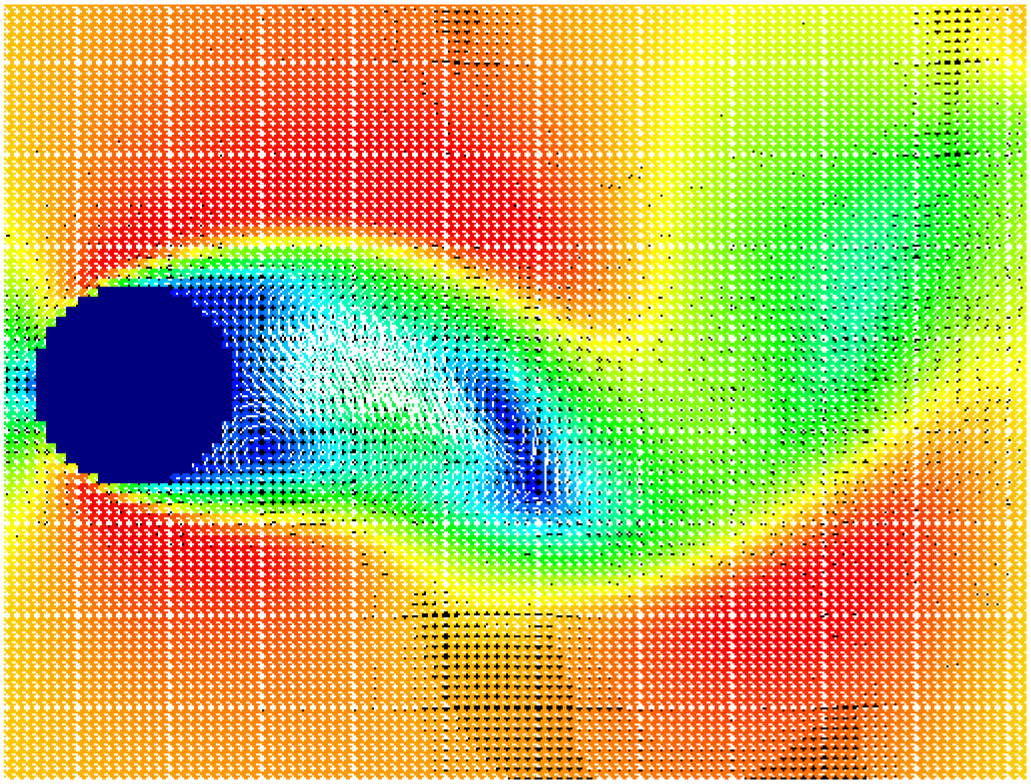


Figure 4.3: Velocity field close-up

In order to quantify these results, I computed the drag coefficient on the cylinder to compare with published data. The drag coefficient was obtained by integrating normal and tangential stresses on the surface of the cylinder and resolving the horizontal component which was then non-dimensionalized based on cylinder diameter. Lift coefficients were computed in a similar manner though resolving the vertical component. Figure 4.4 shows the drag coefficient time history plotted against non-dimensional time for the baseline test.

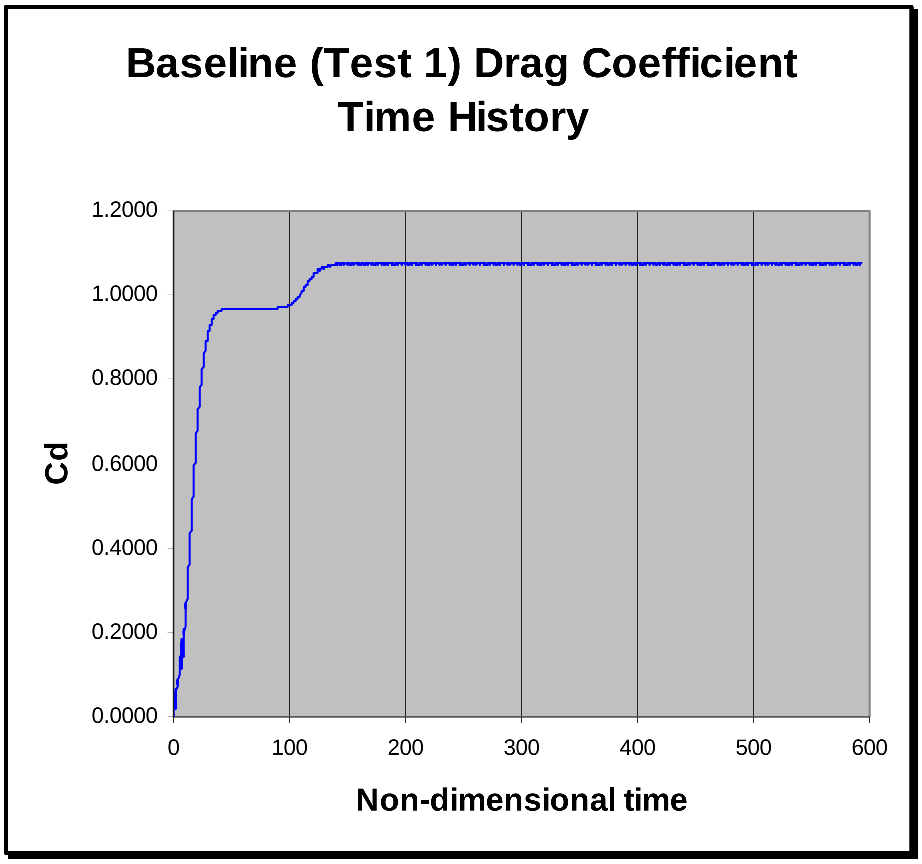


Figure 4.4: Baseline test drag coefficient time history

Figure 4.4 shows that drag coefficient rises steadily as flow velocity is ramped-up. At some point it levels off to a steady value as the trailing vortex pair grows. Once the vortex instability sets in, drag coefficient rises rapidly and starts oscillating at the same time. It then levels off to some mean value but continues to oscillate. Figure 4.5 shows a zoomed-in view of the drag coefficient time history highlighting the oscillating drag coefficient after development of the vortex street.

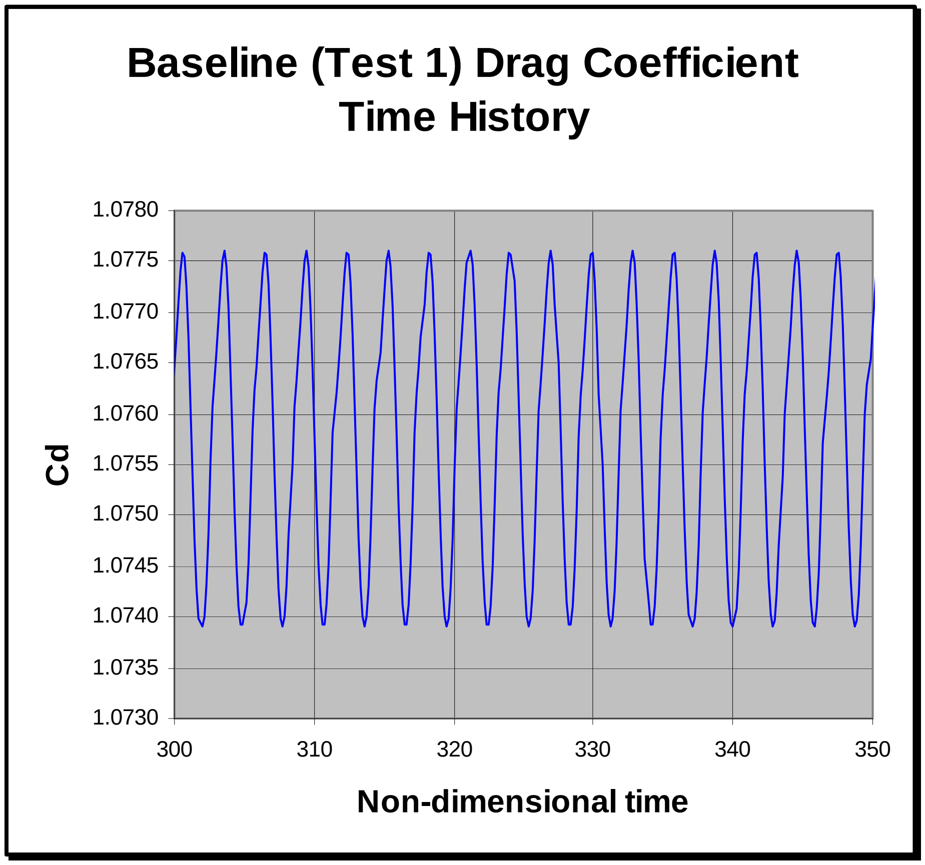


Figure 4.5: Zoomed-in drag coefficient time history

I processed the drag coefficient data using Fourier transform techniques to examine frequency components of the drag coefficient while the vortex street is fully developed. Vortex shedding frequency is half the frequency of oscillation of the drag coefficient. Thus, I was able to compute the corresponding Strouhal number using results obtained from the frequency analysis of drag coefficient.

For the baseline case the computed mean drag coefficient, after vortex street development, was 1.3, which is in good agreement with experimental results (about 1.4). The computed Strouhal number was 0.19, which is a little higher than published empirical results obtained from [76] (0.14 +/- 10%), but is still reasonably close given the uncertainty in the empirical data.

Parametric Tests

Parametric tests at Re = 125 were conducted to check the sensitivity of the baseline results to changes in fluid domain geometry, state equation, artificial Mach number, weighting function, and resolution. For each parametric test, drag coefficient time histories were computed and analyzed and used as the measure of merit to ascertain differences in results between each test. Figure 4.6 shows drag coefficient time histories for a number of parametric tests.

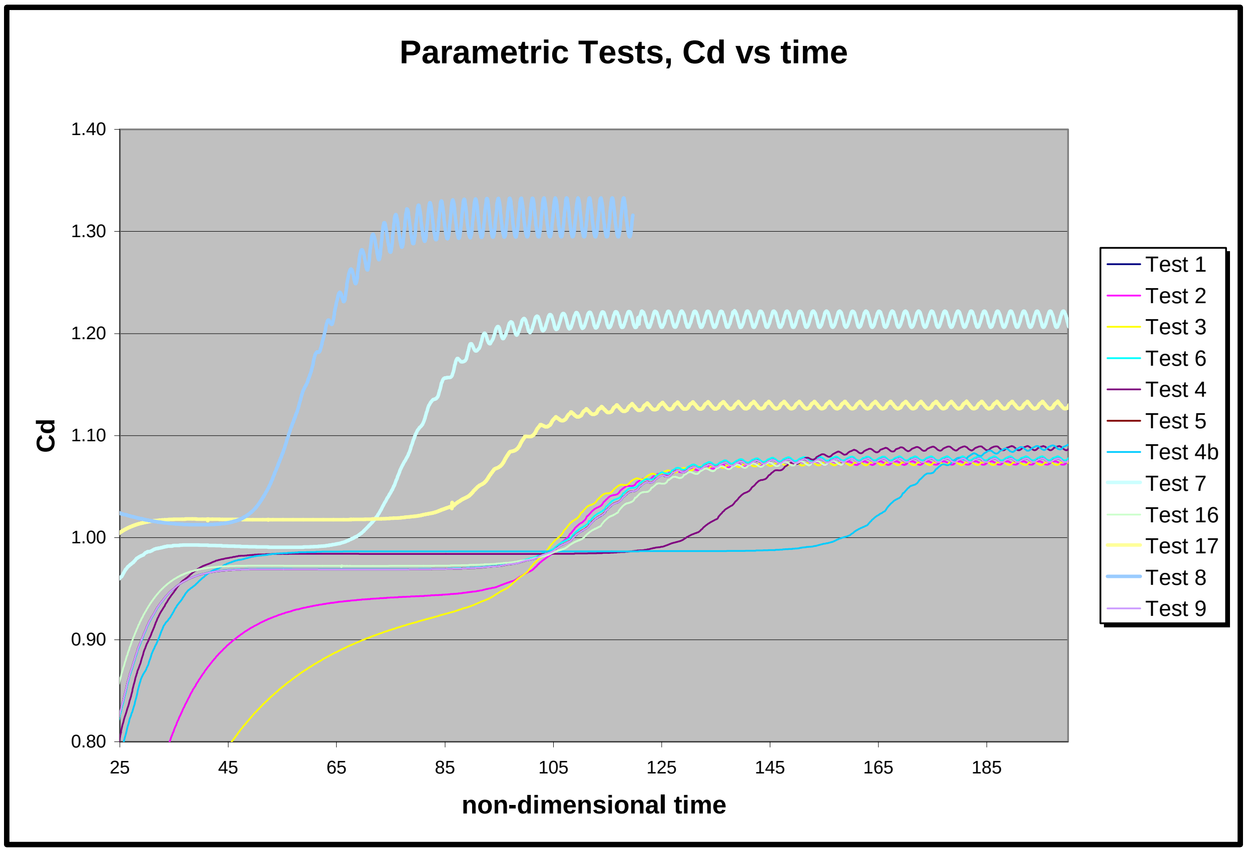


Figure 4.6: Parametric test drag coefficient time histories

In many cases, the drag coefficient time histories cluster towards the baseline test time history (Test 1). However, some tests exhibited marked differences. The following sections discuss these similarities and differences.

Downstream Length Sensitivity

Parametric tests that varied the downstream length of the fluid domain from 10 diameters to 40 diameters showed little variation on the results. Strouhal number and drag coefficient results were virtually unchanged. Figure 4.7 shows the extended downstream fluid domain length along with the velocity field plot clearly showing a well defined vortex street.

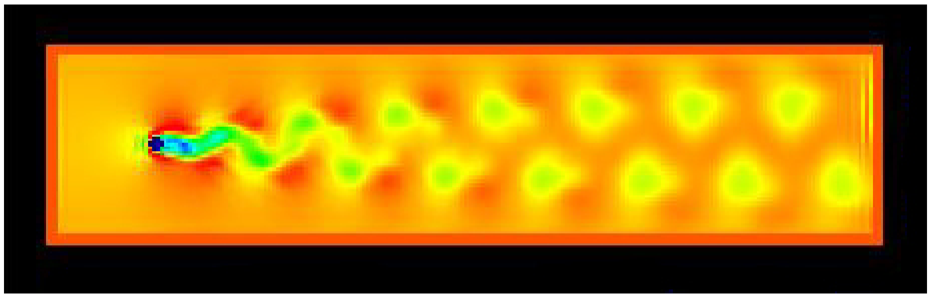


Figure 4.7: Extended domain downstream

The *Test 3* time history shown in Figure 4.6 corresponds to the results shown in Figure 4.7.

Blockage Sensitivity

Tests were also conducted that varied the width of the fluid domain from 5 diameters on either side of the cylinder to 20 diameters on either side. The effect of increasing the width of the domain was to delay inception of the vortex instability. As illustrated in Figure 4.6 baseline results and results from downstream length variation show vortex instability inception to occur around 105 time units (non-dimensional time). However, as the width of the domain increases, inception is delayed from 105 to 130 to 160 time units for baseline, *Test 4*, and *Test 4b*, respectively. The computed drag coefficient changes only slightly.

State Equation Sensitivity

Several forms of a barotropic state equation were tested with negligible variation in the results from the baseline tests. In addition to the state equation shown earlier for the baseline test case, two other state equations were tested:

$$P = c^2 \frac{\rho_o}{\gamma}\left(\left(\frac{\rho}{\rho_o}\right)^{\gamma} - 1\right)$$

and,

$$P = c^2 \rho$$

Weight Function Sensitivity

Similarly, changes in the weighting function from linear to Gaussian showed negligible results from the baseline. The linear weighting function is more computationally efficient and was the one chosen for all further tests.

Artificial Mach Number Sensitivity

Artificial Mach number was varied from 0.3 down to 0.01, with the baseline test using an artificial Mac number of 0.1. The lower the artificial Mach number the more incompressible the simulation approximation. Results of these tests showed little change in results for variations from 0.1 down to 0.05. The lowest artificial Mach number tested of 0.01 did, however, show some variation – vortex instability inception occurred earlier than the baseline and the computed drag coefficient was higher than that computed in the baseline tests. The test run labeled *Test 17* in Figure 4.6 illustrates these results. At higher artificial Mach numbers – 0.2 to 0.3 – the flow was such that the vortex instability never materialized on its own. Disturbances were applied to kick start the instability. Upon release of the disturbances, the vortex street persisted for a short period of time before the flow returned to a stable state with no vortex instability.

Resolution Sensitivity

By far the most dramatic variations from baseline results were due to increasing the resolution of the simulation. Tests were conducted with resolutions from 4,579 nodes (baseline) up to 63,529 nodes. As expected higher resolution tests yielded higher computed drag coefficients which are much closer to the experimentally obtained drag coefficients as published in the fluid mechanics literature. Tests labeled *Test 7* and *Test 8* in Figure 4.6 illustrate some higher resolution results. Figure 4.8 shows *Test 8* results in isolation.

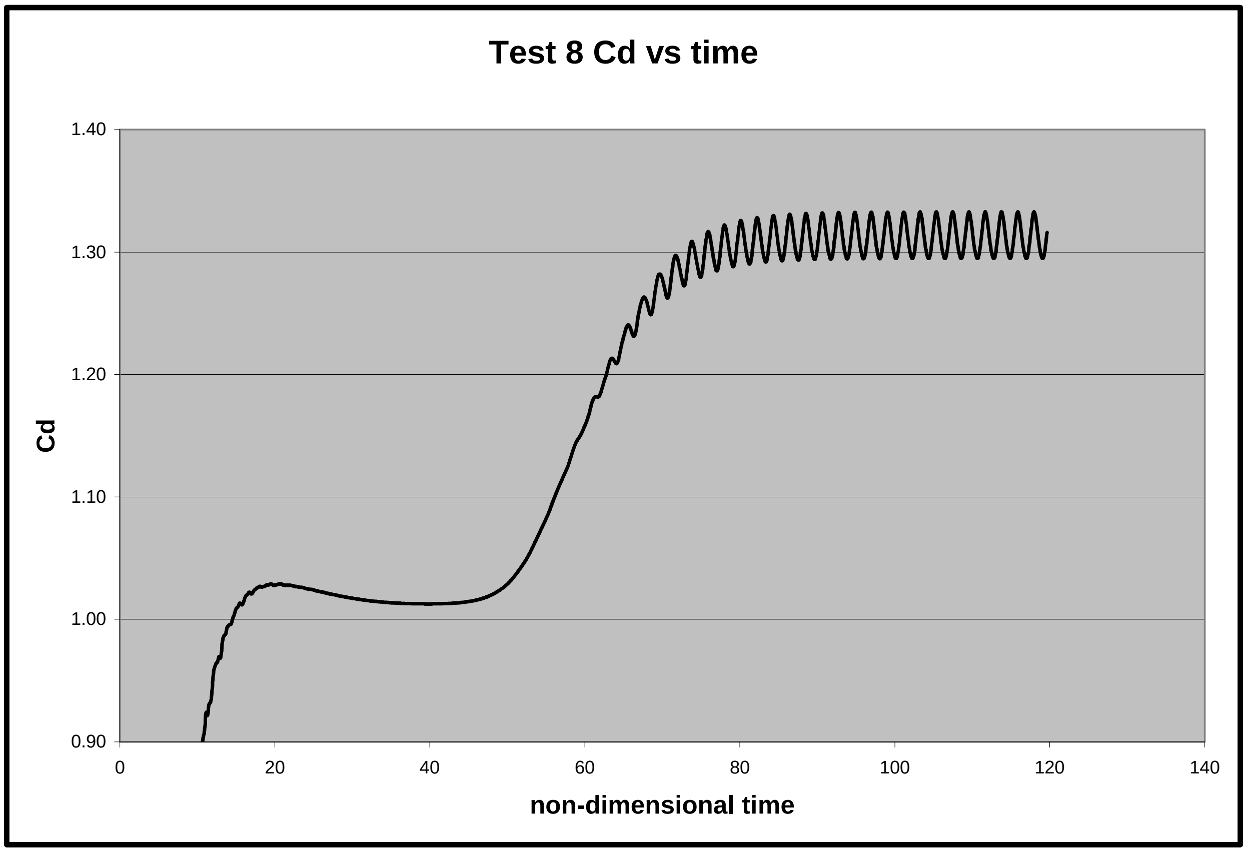


Figure 4.8: Test 8 drag coefficient time history

The drag coefficient obtained from *Test 8* was 1.3, which compares well with published experimental results of approximately 1.4 at a Reynolds number of 125.

Higher resolution tests also showed greater range in drag coefficient as it oscillates about the mean drag coefficient post-vortex instability inception. Further, vortex instability inception was found to occur earlier for higher resolution tests.

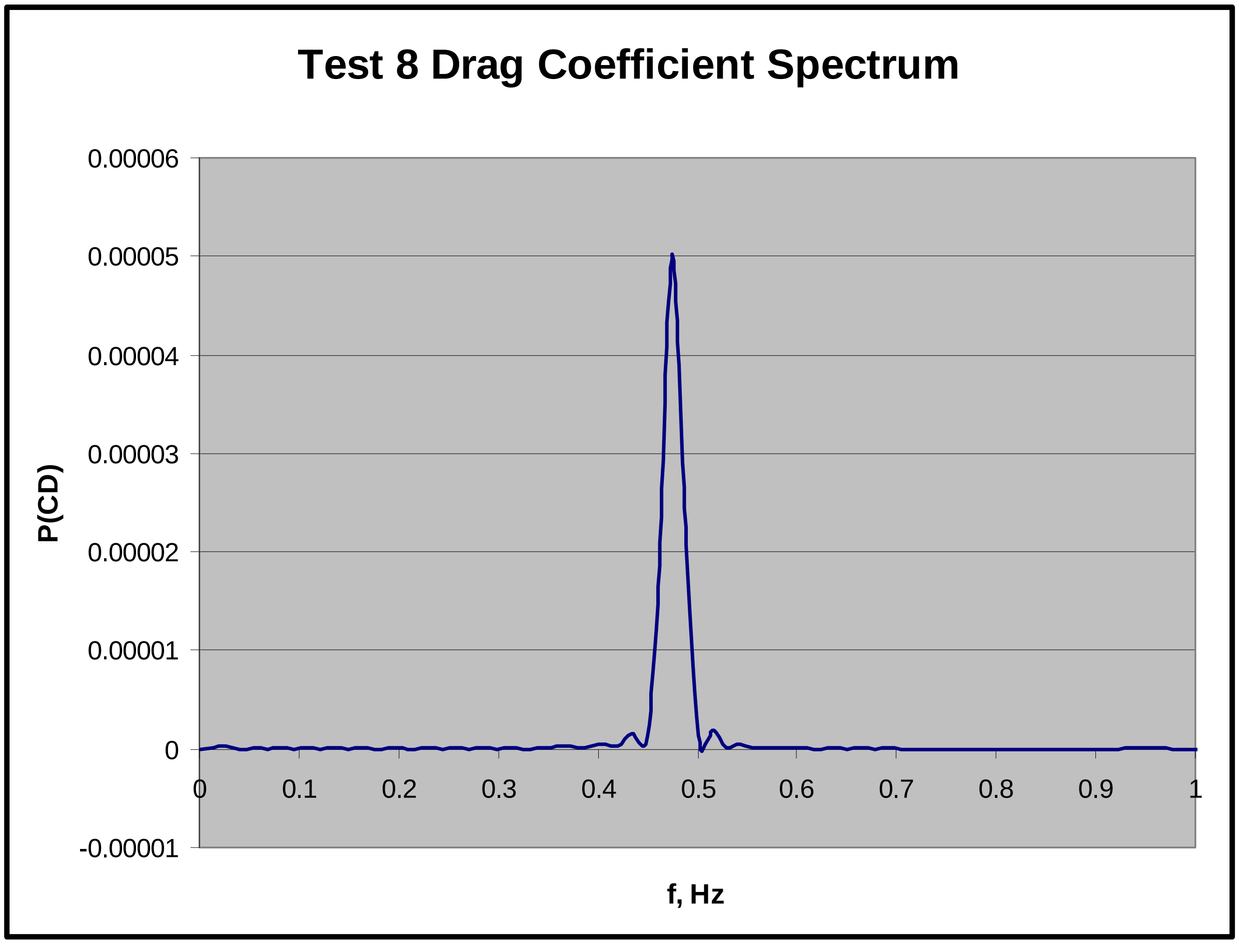


Figure 4.9: Test 8 drag coefficient power spectrum

Figure 4.9 shows the drag coefficient power spectrum for Test 8. The computed Strouhal number for this test was 0.19.

Reynolds Number Test Results

I conducted several additional tests over a range of Reynolds numbers to see how well the DPM performs. I chose a range of Reynolds numbers from 2.5 to 1000. Flow past circular cylinders over this range is very well understood with many available sources of qualitative and quantitative information on such flow.

At Re = 2.5 DPM results show unseparated flow as was expected from published experimental data. The computed drag coefficient was 4.3 in this case.

At Re = 10 experimental results suggest that we should start to see a fixed pair of vortices in the wake of the cylinder. Indeed, DPM simulations show this behavior. The computed drag coefficient in this case was 2.1.

Literature suggests that at Re = 65 we might see development of a laminar vortex street. DPM tests at this Reynolds number were, however, stable and did not develop a vortex street. A vortex pair was developed but remained stable. Tests were run where the flow was purposefully disturbed to instigate vortex instability and development of a vortex street. What happened was that after a disrupting impulse was applied a vortex street did indeed develop; however, after some time the flow re-stabilized reverting back to a steady vortex pair wake structure. The drag coefficient obtained in this case was 1.1; however, this is expected to be low given the lack of development of a vortex street.

At a Re = 125, which was that used for our baseline tests, we should expect to see development of a laminar vortex street. As discussed earlier, vortex streets were consistently developed with clearly defined flow structures in the wake. Tests at this Reynolds number yielded a drag coefficient of 1.3 and a Strouhal number of 0.19.

The highest Reynolds number tested was at Re = 1000. At this Reynolds number, literature suggests development of a turbulent vortex street. So far I have not included a turbulence model in the DPM but the results of this test show some turbulent characteristics. In addition to exhibiting a dominant vortex shedding frequency, the drag coefficient data (over time) exhibited oscillations at other frequencies some higher than, some lower than, and others at harmonics of the vortex shedding frequency. The computed drag coefficient in this case was 0.98 and the computed Strouhal number was 0.2, which agrees well with published experimental results of 0.21 +/- 10%.

The fact that the current implementation of DPM does not include an explicit turbulence model seems noteworthy given the accuracy with which Strouhal number and drag coefficient were computed in this case. At this Reynolds number, experimental data indicates the wake behind the cylinder should be turbulent. Reference [77] presents results of a study of the unsteady flow around a circular cylinder using a two dimensional finite volume method incorporating three different turbulence models.

For the case of Reynolds number equal to 1000, [77] reports predicted Strouhal numbers of approximately 1.5, 1.7, and 2.8 using standard k-epsilon, realizable k-epsilon, and SST K-omega turbulence models, respectively. The reported experimentally obtained Strouhal number is approximately 0.21. The DPM predicted Strouhal number discussed herein is 0.2, which is much closer to the experimental value compared to the turbulence model predicted values. Moreover, the predicted drag coefficient using the present DPM method matches one of the turbulence model predictions and is closer to the experimental drag coefficient than two other predictions reported in [77]. Some researchers [29] report inherent turbulence capturing capabilities in SPH. I have to wonder if the DPM method implemented here is showing similar characteristics. While not the subject of this dissertation, I believe this turbulence capturing aspect of DPM, whether found to be true or not, deserves further investigation.

Turning back now towards the present Reynolds number tests, Figure 4.10 shows a plot of computed drag coefficient versus Reynolds number over the range of Reynolds numbers discussed earlier. Also shown for comparison is a curve of published experimentally obtained drag coefficients.

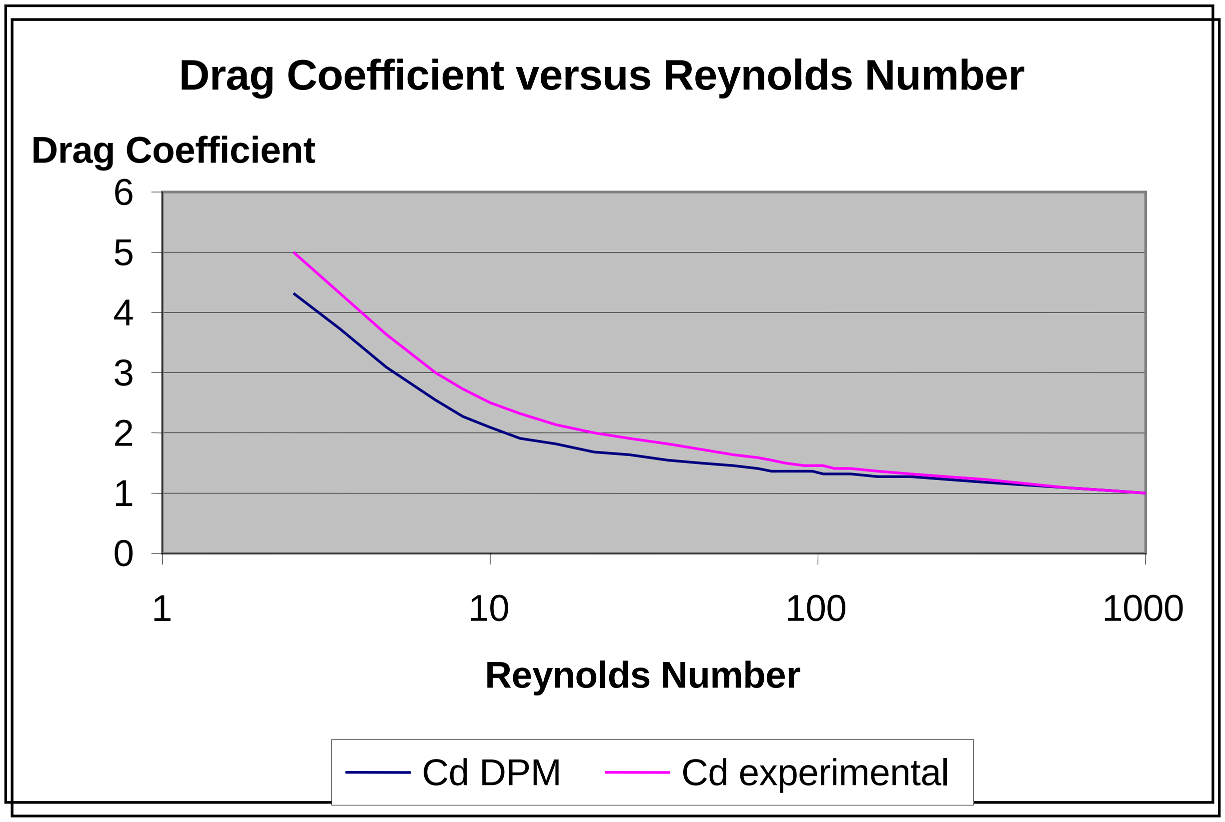


Figure 9: Drag coefficient results

The DPM predicted drag coefficients agree fairly well with the experimental results. The DPM results somewhat under predict drag coefficient at lower Reynolds numbers. The computed and experimental drag coefficient curves converge at higher Reynolds numbers.

Chapter 5

EXTENSION TO MULTIPHASE FLOW

So far this dissertation has focused on the single, liquid phase implementation of DPM. At the outset of this research the aim was to investigate cavitation. This requires modification of the DPM to deal with multiphase fluid flow, specifically in this research, where the flow is comprised of fluid in a liquid phase – liquid water – and fluid in a vapor phase – water vapor. Modifications to DPM consist of extending the current method rather than developing separate framework to deal with cavitation.

Cavitation is characterized by a pressure-density relationship where when the pressure in a flow is reduced to a critical level, below the vapor pressure, the fluid changes phase with a corresponding change in density from that of liquid to that of vapor. (In this dissertation, the fluid is always water and references to liquid and vapor correspond to liquid water and water vapor, respectively.) The governing equations used in DPM consist of the momentum equations, the continuity equation, and a barotropic state equation. The state equation represents an explicit relationship between density and pressure. Given the character of cavitation it is clear that this state equation should be the center of attention while extending the method to cavitating flow.

The approach taken herein is to maintain the single-fluid nature of DPM in that both liquid water and water vapor will be treated as a single fluid subject to the same governing equations rather than treat them as two separate fluids subject to their own sets of governing equations. This approach requires a state equation that's capable of modeling the phase transition between liquid and vapor. Such as state equation has been developed as is discussed in this chapter. The multiphase state equation developed in this research is based on a few key assumptions:

- The flow is isothermal and a barotropic state equation is used.
- Cavitation inception occurs when the local pressure drops below the fluid's vapor pressure. No consideration is given to other important factors affecting cavitation inception such as entrained cavitation nuclei.

That said, there are some key assumptions that are not made as part of this development:

- A cavity closure model is not required as in the case of potential flow based cavitation modeling.
- Cavitation inception locations need not be specified or represented with sharp corners.

Single Phase State Equation

Before going into the details of the multiphase model, it will be helpful to review in more detail the single phase state equation used in DPM so that ready comparison and contrast may be made. This review will also serve to illustrate why a new state equation is required to handle cavitation. The single phase state equation adopted in DPM is as follows,

$$P = c^2 \frac{\rho_o}{\gamma}\left(\frac{\rho}{\rho_o}\right)^{\gamma}$$

Here, *c* is an artificial sound speed used to enforce incompressibility and γ is the specific heat ratio of the fluid. ρ_o is a reference density, and ρ is the actual density at a given location in the fluid. This state equation is a standard barotropic state equation where pressure is an explicit function of density only. Temperature does not appear in this state equation and the flow is considered isothermal.

In single phase fluids, where *c* is held constant, the term, $c^2 \frac{\rho_o}{\gamma}$, is constant and represents a reference pressure. The only variable in this state equation is ρ. Therefore, *P* at any given point in the fluid is simply a function of the density ratio, ρ/ρ_o at that point. If ρ equals ρ_o then *P* is equal to the reference pressure. Figure 5.1 shows a graph of this equation using a ρ_o of 999.5 kg/m^2, a γ of 7, and a *c* corresponding to a Mach number of 0.05.

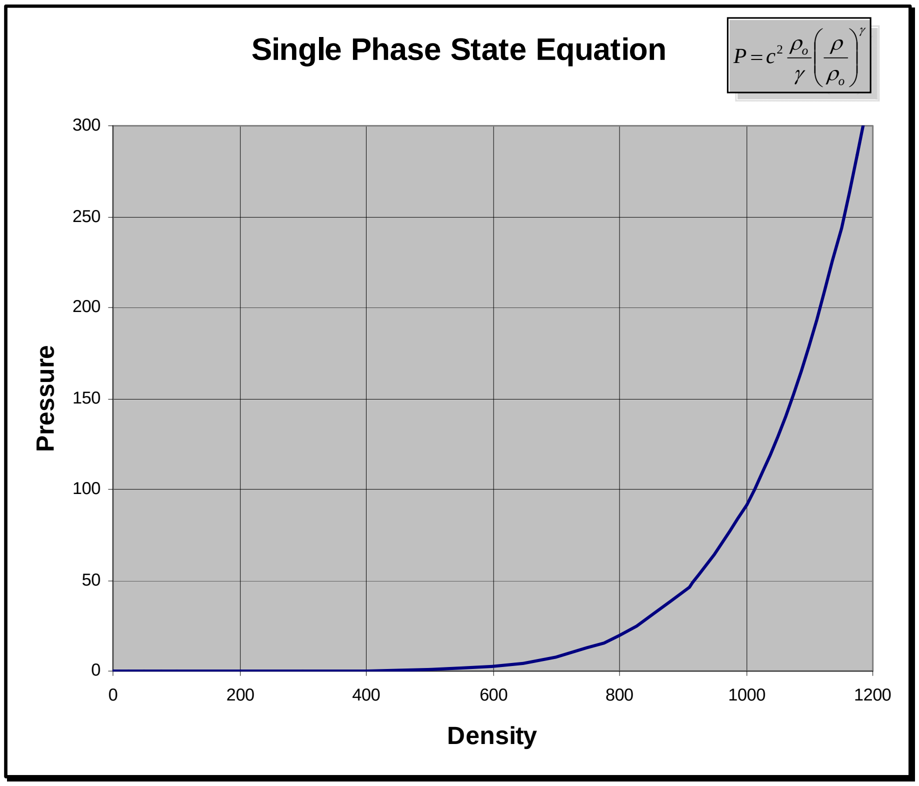


Figure 5.1: Barotropic state equation graph

For incompressible flow, this equation and the selection of c is designed to keep ρ approximately equal to ρ_o. If ρ drops below ρ_o, as governed by the momentum and continuity equations, then there will be a disproportionately large decrease in pressure. One can imagine this large reduction in pressure will act to draw fluid particles back into the low density region thereby causing the density to increase. As ρ increases, P increases disproportionately tending to push fluid particles apart in order to bring ρ back down towards the reference density. This back and forth variation in density around the reference density is a manifestation of the slightly compressible nature of the current method.

This behavior is precisely what is desirable to model incompressible flow. One can vary c to control the degree to which P changes with respect to changes in density. Figure 5.3 shows graphs of this state equation for various values of c shown as various values of Mach number.

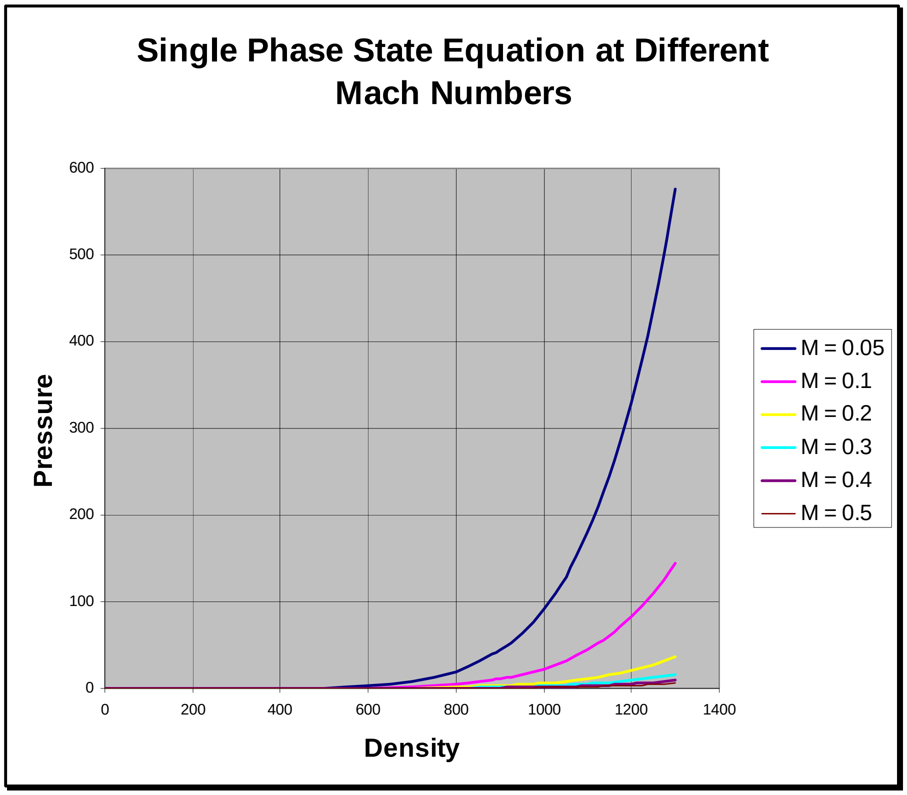


Figure 5.2: State equation graphs for various Mach numbers

In order to approximate an incompressible fluid the artificial Mach number must be at least less than 0.3 and preferably around 0.1. The higher the Mach number the greater the compressibility of the fluid. As Mach number equals the characteristics reference velocity for the problem at hand divided by the sound speed, a suitable artificial sound speed should be selected to yield a Mach number within the desirable range.

The effect c has on the state equation is clear upon examining the graphs in Figure 5.2. Specifically, it is the slope of the state equation curves at the reference density that governs the incompressibility. For the $M = 0.5$ case, the slope is very high such that small deviations in density from the reference density result in very large changes in pressure. At the other extreme, the curve corresponding to $M = 0.2$ shows a much more gentle variation in pressure with respect to density deviations. As stated earlier it is the disproportionate change in pressure that keeps the density from straying too far from the reference density when modeling incompressible flow.

One might be tempted to select very large values for *c* in order to ensure incompressibility; however, serious numerical instabilities would result. Therefore, *c* must be tuned to yield the desired level on incompressibility without causing numerical instabilities. Monaghan [11] recommends, for SPH, that *c* be selected according to the relation,

$$c = \max\left[\left(\frac{V_o^2}{\delta}\right), \left(\frac{\upsilon V_o}{L_o \delta}\right), \left(\frac{F L_o}{\delta}\right)\right]$$

where $\delta = \frac{\Delta\rho}{\rho_o}$. All of the subscripted parameters in the above expression represent reference values characteristic of the problem under investigation. δ represents the allowable percentage change in density relative to the reference density. The lower the allowable percentage change in density, the higher will be *c*. I have used this same criteria in DPM with success. (The circular cylinder tests from the previous chapter explored the method's sensitivity to *c* via changes in Mach number.)

The state equation discussed so far is quite suitable for incompressible flow so long as the caveats on *c* are adhered to. Using low values for *c* would make this equation suitable for some compressible flow problems as well. However, this equation is not suitable for modeling the liquid vapor phase transition characteristic of cavitation. For that we need an alternative state equation.

Multiphase State Equation

The phase transition at the point of cavitation is characterized by a large decrease in density at a constant pressure. For our purposes this pressure is the vapor pressure. The single phase state equation does not exhibit this behavior unless the pressure falls to zero in which case the density does also (see Figure 5.1). What we really want is a state equation that looks like that shown in Figure 5.3.

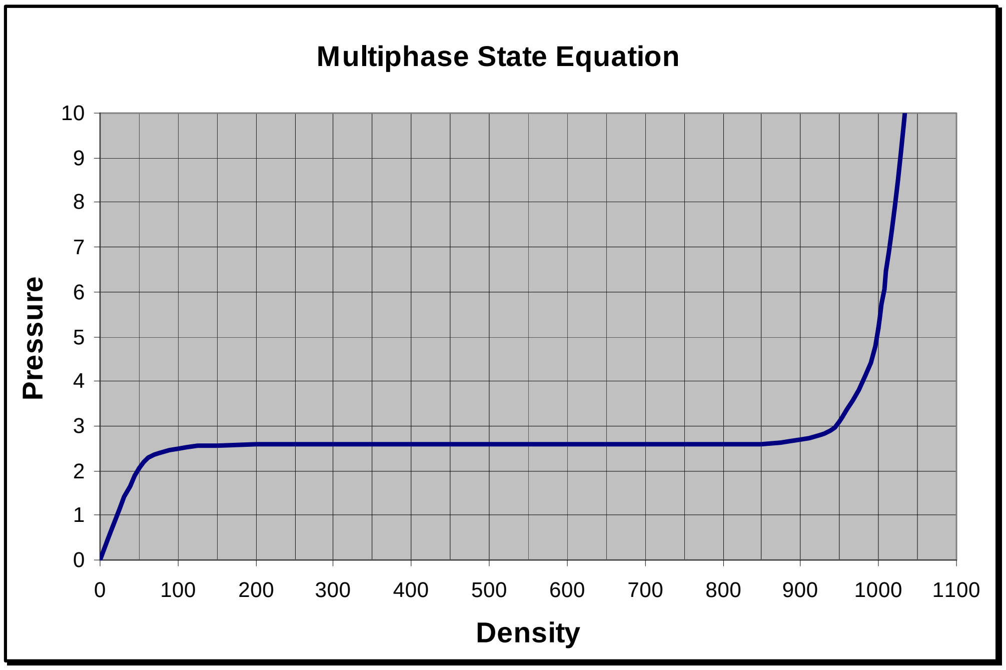


Figure 5.3: Multiphase state equation

Before examining the equations corresponding to this graph it is useful to examine the shape of the graph first. If you look at the part of the curve around the density of liquid water, approximately 1000 kg/m^3, you can see that the curve resembles the single phase state equation in that small changes in density result in relatively large changes in pressure. This same characteristic is evident on the other end of the density spectrum as the density approaches that of water vapor. This behavior is important so as to model each phase as nearly incompressible. However, when the pressure reaches some critical pressure this curve shows a dramatic reduction in density at a constant pressure. This is the point of phase transition.

Another way of looking at this behavior is to consider the effective speed of sound, c_e, corresponding to this state equation. For isentropic processes the sound speed is defined as,

$$c_e^2 = \frac{dP}{d\rho}$$

I have used the subscript on c to distinguish this sound speed from the artificial sound speed discussed earlier. This equation states that the sound speed is proportional to the rate of change of pressure with respect to density, or in other words, the slope of the line representing the state equation. For the single phase state equation discussed in earlier chapters, $P = c^2\rho$, c^2 is simply the slope of the straight line represented by this equation. For the other single phase state equations and the multiphase one discussed herein, the slope is not constant implying that the sound speed is not constant. Examination of the graph in Figure 5.3 suggests that the slope, and thus the effective sound speed, near the reference density is relatively large. Likewise, the slope and thus the effective sound speed near the vapor density is large. However, the slope over the transition density range is zero (or nearly so as will be seen shortly) indicating a sound speed of zero (theoretically infinite Mach number) which is consistent with the instantaneous reduction in density at the critical pressure.

Figure 5.4 shows the effective sound speed corresponding to the state equation graph shown in Figure 5.3.

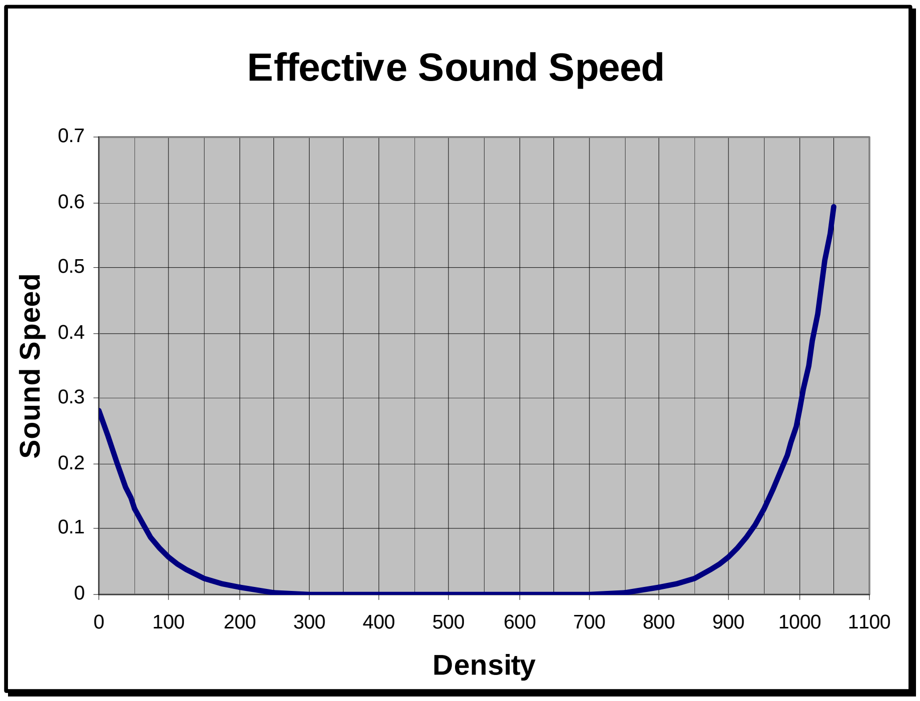


Figure 5.4: Effective sound speed

This curve is simply a graph of the derivative of the curve shown in Figure 5.3. Here you can see that the effective sound speed is relatively high towards the ends of the curve where there exist distinct phases – liquid and vapor. The flat portion in the middle represents a low effective sound speed. Another way to visualize what's happening here is to examine the graph of the effective Mach number. Figure 5.5 shows such a graph corresponding to the effective sound speeds plotted in Figure 5.4.

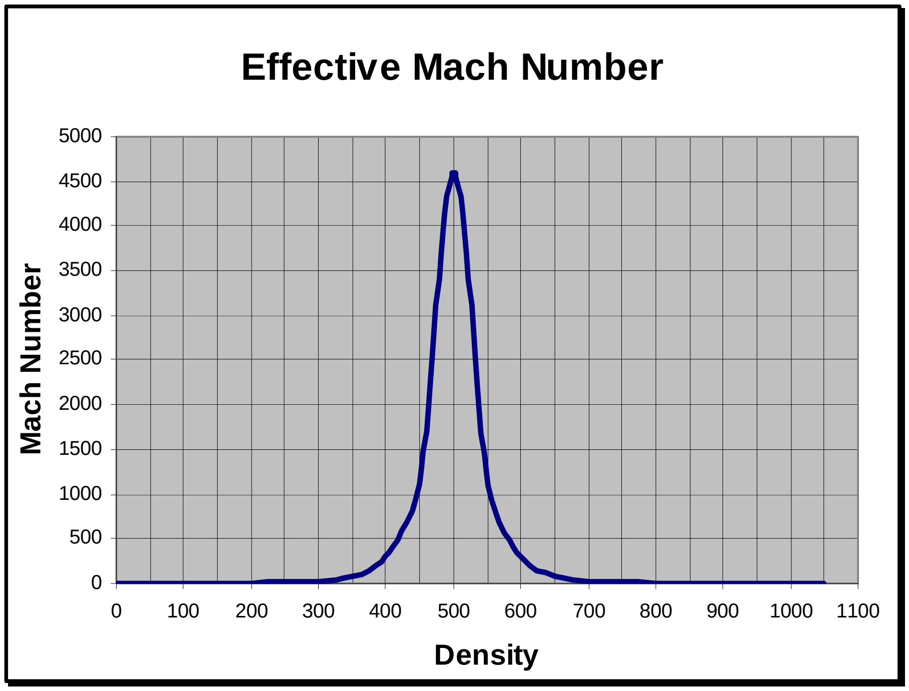


Figure 5.5: Effective Mach number

This spike in Mach number is clearly evident and shows where the fluid will behave in a compressible manner corresponding to the phase transition.

The Equations

So far I have only discussed the character of the multiphase state equation. Now, I'll turn attention to the equation expression itself. There are many expressions one may devise to represent a curve such as the one shown earlier in Figure 5.3. Some forms can be derived using the logistic function or transcendental functions. I started with the single phase state equation since it already exhibited the desirable characteristics for modeling each phase as an incompressible fluid. (Treating each phase, including the vapor phase, as incompressible, or nearly so, is justified since all the flows considered in this research are restricted to low Mach number flows.) Basically, I added terms in the single phase state equation in order to compress it vertically so as to raise the pressure at which the density decreases rapidly to that of the a given vapor pressure. I then developed another part of the equation that effectively mirrors this first part about both the vertical and horizontal axes. This resulted in a piecewise continuous state equation containing a new variable, *s*, I call a *cavitation factor*. The resulting expressions for the multiphase state equation are:

$$P = \begin{Bmatrix} \dfrac{c^2\rho_o}{\gamma}\left[\left(\dfrac{\rho}{\rho_o}\right)^{\gamma}(1-s)+s\right] \; ; \; if \quad \rho \geq \dfrac{(\rho_l - \rho_v)}{2} \\ \dfrac{c^2\rho_o}{\gamma} s\left[\left(\dfrac{\rho-\rho_o}{\rho_o}\right)^{\gamma}+1\right] \; ; \; if \quad \rho \leq \dfrac{(\rho_l - \rho_v)}{2} \end{Bmatrix}$$

Here, ρ_l and ρ_v are the reference liquid and vapor densities, respectively. The cavitation factor, s, is defined as follows,

$$(1-s) = \frac{(P_o - P_v)}{P_o}$$

Essentially, *(1-s)* represents the percent difference between the reference pressure and the vapor pressure at which cavitation is assumed to occur. The greater this percent difference, the less propensity for the fluid to cavitate, which is the same as saying the cavitation number in the traditional sense is high. Conversely, the smaller this percent difference, the greater the propensity for the fluid to cavitate, which is the same as saying the cavitation number is low. We can relate the traditional cavitation number, σ, to the cavitation factor, *s*. Cavitation number is defined as,

$$\sigma = \frac{P_o - P_v}{\frac{1}{2}\rho_o V_o^2}$$

Rearranging the definition of cavitation factor to isolate *$(P_o$-$P_v)$* yields,

$$(1-s)P_o = (P_o - P_v)$$

which can be substituted into the definition for σ to arrive at,

$$\sigma = \frac{P_o(1-s)}{\frac{1}{2}\rho_o V_o^2}$$

Now the reference pressure, P_o, is equal to $\frac{c^2\rho_o}{\gamma}$ and substituting this into the previous expression yields,

$$\sigma = \frac{2c^2(1-s)}{\gamma V_o^2}$$

However, the Mach number, *M*, is defined as $\frac{V_o}{c}$. Squaring this term, inverting, and substituting it into the previous expression yields,

$$\sigma = \frac{2(1-s)}{\gamma M^2}$$

This expression provides a simple means of computing the characteristic cavitation number for a given problem when *s* is specified. Or, this expression can be used to compute the required *s* for use in the state equation corresponding to a desired cavitation number.

Discussion of Parameters

The expression for the multiphase state equation contains a couple of parameters worth discussing further, namely, γ and *s*.

Earlier I mentioned that γ is the specific heat ratio usually taken as 7 in the single phase state equation. Using a value of 7 in the multiphase state equation yields a graph with the shape shown in Figure 5.6.

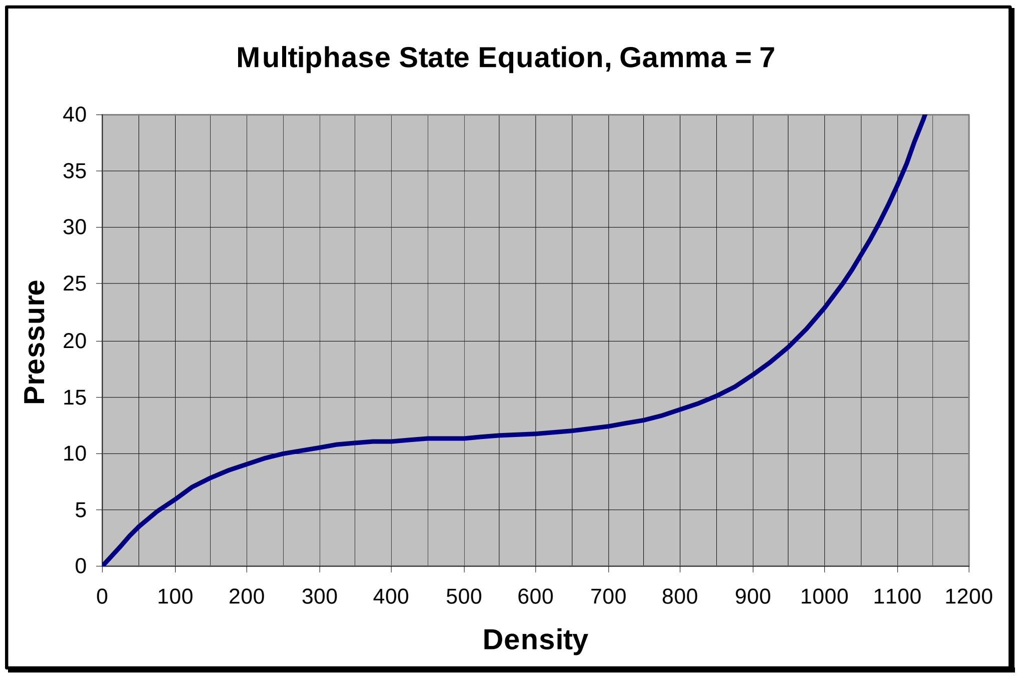


Figure 5.6: Multiphase state equation with $\gamma = 7$

The shape of this curve is different from that shown in earlier in Figure 5.3 in that the phase transition is somewhat more gradual. This shape is just fine for simulating multiphase flow, however, it has the effect of smoothing the phase interface – the boundary between liquid and vapor in a simulation. The shape of the curve shown in Figure 5.3 yields a shaper phase interface. In order to achieve the shape shown in Figure 5.3 you must use a value for γ greater than 7. A value of 31 was used for the graph shown in Figure 5.3. Doing this deviates from the definition of γ as the specific heat ratio and relegates γ to a tuning factor used to control the shape of the state equation. In general, the higher the value for γ the shaper the phase transition. The only restriction here is that γ must be an odd number; otherwise, the shape of the curve at lower densities will be mirrored about the horizontal axis.

It's also important to note that γ has an effect on the value of the reference pressure and it should be understood that pressures shown on the graphs and determined by the state equation are essentially scaled pressures. This does not present difficulties in general since the magnitude of the pressure is not so important as is the relative magnitudes of pressure and pressure gradients. One can scale pressure to any desired reference pressure by developing a suitable scale factor.

I've already discussed *s* and showed how it relates to the traditional cavitation number. However, it is worth looking at several examples of how the value of *s* affects the shape of the multiphase state equation. Figure 5.7 shows graphs of the state equation corresponding to different values for *s*.

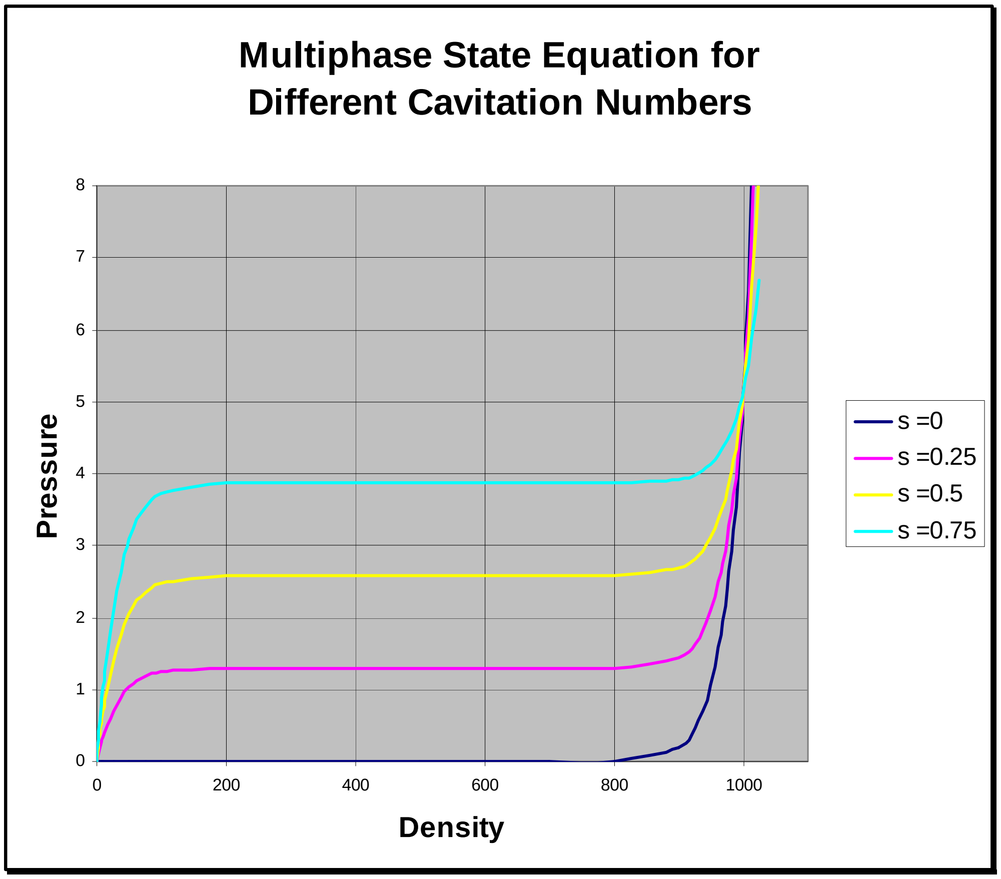


Figure 5.7: Effect of cavitation factor

In general, as *s* gets larger, the cavitation number goes down, which implies that the difference between the reference pressure and the vapor pressure is small. *s* basically has the effect of shifting the horizontal part of the state equation curve – the phase transition part – up or down.

A value of zero for *s* corresponds to single phase flow as indicated by the $s = 0$ curve in Figure 5.7. Thus, this same multiphase state equation can be used to model incompressible flow by setting γ to 7 and *s* to 0. To model cavitating flow, one can tune γ and set *s* to a value corresponding to the desired cavitation number using the expression relating cavitation number to s shown earlier.

Chapter 6

CAVITATING FLOW NUMERICAL EXPERIEMENTS

In order to test the multiphase state equation's ability to model cavitation, I embarked on several numerical experiments with several aims:

- I wanted to assess if and how well the multiphase state equation captures cavitation for low cavitation number flow about several different geometries.
- I wanted to achieve high density ratios of 1000:1.
- I wanted to gain some insight into what happens in the cavity closure region for unsteady cavities.

Over the course of this research, many successful and not so successful simulations were run. The not so successful simulations where those that didn't achieve density ratios of 1000:1 or were unstable to point of crashing the simulation. Through trial and error, code revisions, and tuning successful simulations were made that achieve the set objectives. The following sections describe several representative result sets for three different problems.

Circular Cylinder Tests

The first series of cavitation tests performed using the DPM were of cavitating flow past a circular cylinder. This problem was a natural extension of the series of circular cylinder tests discussed earlier. The primary aim of these first cavitation tests was to see if the cavitation model, i.e., the state equation and approach described in the previous chapter, would actually work. A secondary aim was to see if the model could capture cavitation inception around or behind a round object with no distinct corners. Recall from the previous chapter that no assumptions are made with regard to separation points or cavity shapes.

The geometry and set up of this problem is identical to the circular cylinder tests discussed in Chapter 4. The only difference is the use of the multiphase state equation. Referring to the multiphase state equation presented in the previous chapter, these first tests were carried out using a γ equal to 7. Figure 6.1 shows some typical results. This particular image is of the density field for Reynold's number of 10^5 and a cavitation number of 0.7. This image was taken approximately 0.1 seconds after the simulation was impulsively started and it illustrates the formation of the cavity behind the cylinder.

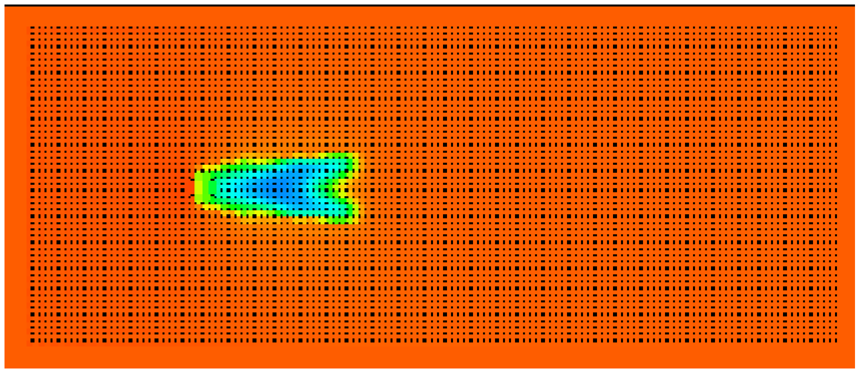


Figure 6.1: Cavity formation behind circular cylinder

Note the shape of the cavity trailing edge. This V-shape was typical of all circular cylinder tests as the cavity took shape. After another 0.1 seconds, the cavity reached the shape shown in Figure 6.2.

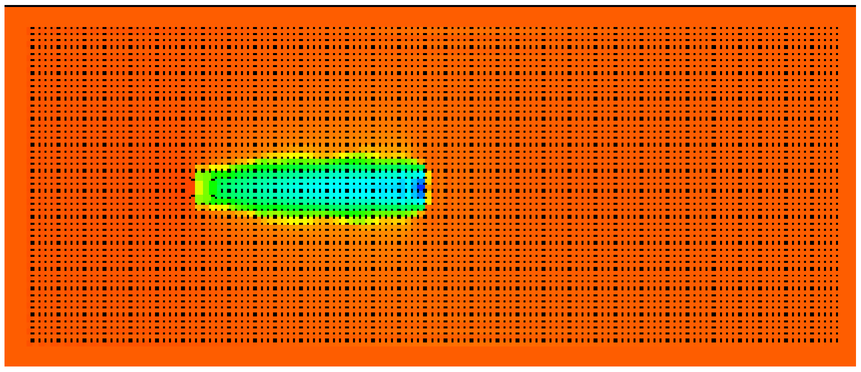


Figure 6.2: Formed cavity behind circular cylinder

The cavity retained this basic shape and length of approximately 5.5 cylinder diameters, however, it was not steady. Very shortly after assuming the shape shown in Figure 6.2, a small bubble became pinched off the trailing edge of the cavity. Figure 6.3 illustrates this phenomenon.

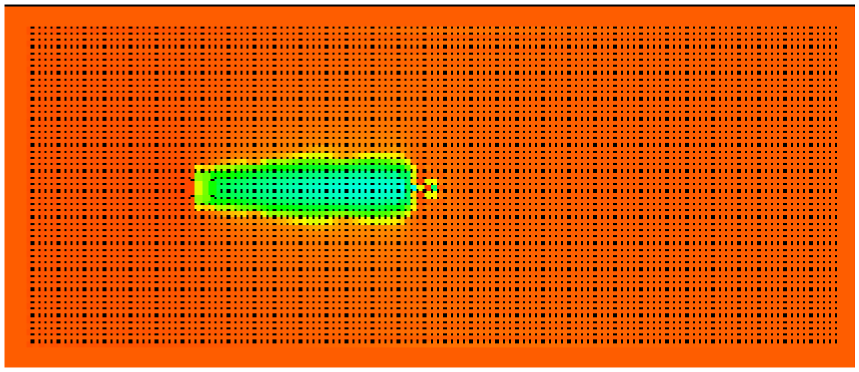


Figure 6.3: Pinched bubble at cavity trailing edge.

At the same moment corresponding to the density field in Figure 6.3, the pressure field showed a small region of relatively high pressure near the pinched bubble. Figure 6.4 shows the pressure field.

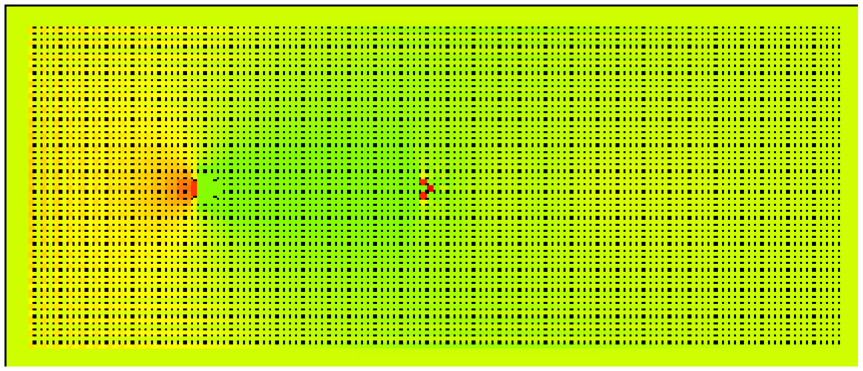


Figure 6.4: Pressure field at instant of pinch

Unfortunately, the color scale was set such that it was difficult to discern the relatively low pressure region corresponding to the cavity. Nonetheless, the red nodes on the right illustrate the high pressure at the pinch point. The velocity field associated with this instant is shown in Figure 6.5.

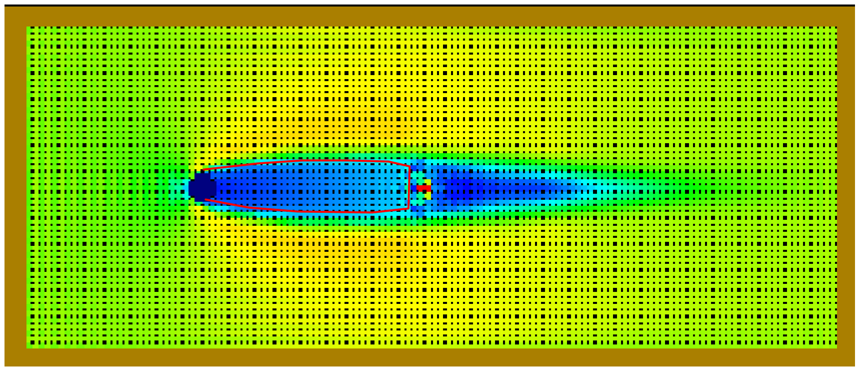


Figure 6.5: Velocity field at instant of pinch

This illustration shows the velocity magnitude where warmer colors represent higher speed. This picture is typical of all circular cylinder cavitation tests where the velocity in the cavity region is relatively low and where there exists a wake region of non-vapor aft of the cavity. The cavity is outlined in red in Figure 6.5. This wake resembles the wake behind a non-cavitating circular cylinder prior to development of the vortex instability. Figure 6.6 shows an example.

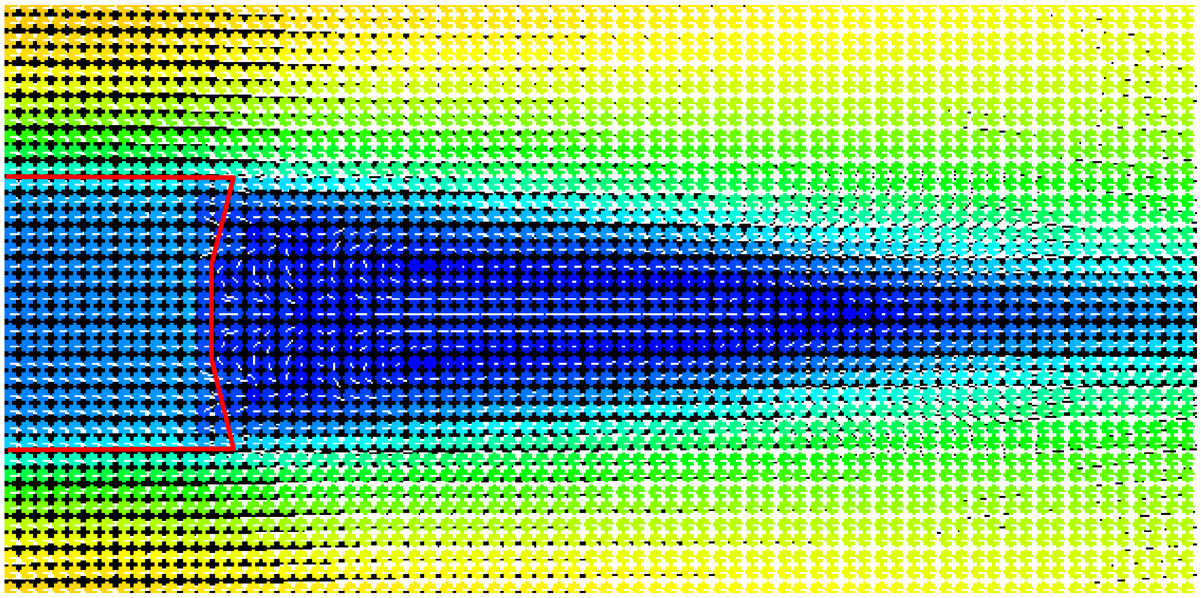


Figure 6.6: Close-up of velocity field

The image in Figure 6.6 includes white tick marks at each node that indicates the direction of flow. Here again, the cavity region is approximately outlined in red.

Along the centerline of the wake region there exists a jet of flow towards the cavity, i.e., in a direction opposing the free stream flow. As this flow approaches the aft end of the cavity it appears to push against the cavity and is redirected outward and then aft in the free stream flow direction. This patter resembles the stretched eddies that form behind the cylinder in incompressible low Reynold's number flow.

This region where the jet in the wake meets the cavity, where the bubble pinches off, proved difficult to capture in these tests. In all tests conducted with the cavitating circular cylinder the simulation would crash shortly after the bubble pinched off. The crash manifested itself as the velocity magnitude and pressure in the pinch region growing uncontrollably. The solution to this issue turned out to be the incorporation of a different numerical integration scheme as will be discussed later in this chapter.

Leading up to this cavity closure instability, the simulation worked fairly well – the multiphase state equation was able to model the inception and growth of the cavity. There was, however, one issue with the cavity that was puzzling at first and that is that the density within the cavity never reached the magnitude of vapor density. The reference density was 999.5 kg/m^3 and the lowest density within the cavity was on the order of 50 kg/m^3. Further, the interface between liquid and vapor was not sharp. The solution to this problem turned out to be to simply use a higher value for γ than 7 as is discussed in the next section.

Flat Plate Tests

As stated in the previous section, density in the cavity behind the circular cylinder never quite reached the vapor pressure. The pressure ratio achieved was on the about 20:1, but not the 1000:1 ratio I was aiming to achieve. In order to troubleshoot the issue, I set up a simple model of flow past a rectangular plate. This problem was selected because it is well understood how and where cavities are likely to form and I wanted to remove surface curvature from the equation, so-to-speak.

In these tests the flat plate was 1 unit in length and ¼ a unit in thickness oriented with its long dimension perpendicular to the flow. The fluid flows past the plate from left to right. Tests at a Reynold's number of 2000 were conducted. At first these test results were similar to the circular cylinder in that the density within the cavity never quite reached vapor density. After some trial and error, I found that using a γ of 31 resulted in the achievement of vapor density within the cavities. This is by virtue of sharpening of the pressure versus density curve representing the multiphase state equation as discussed in the previous chapter.

Figure 6.7 shows some typical results obtained using higher values for γ. This particular image shows the density field. At the start of this simulation the velocity was ramped up to steady state. Once there, the cavitation number, σ, was lowered over several time steps to approximately 1.3 where the snapshot in Figure 6.7 was captured. The rectangular plate is highlighted in Figure 6.7. Also, cooler colors represents lower density. The ambient reference density (liquid water) is shown in orange and the green/cyan colors indicate the forming cavities.

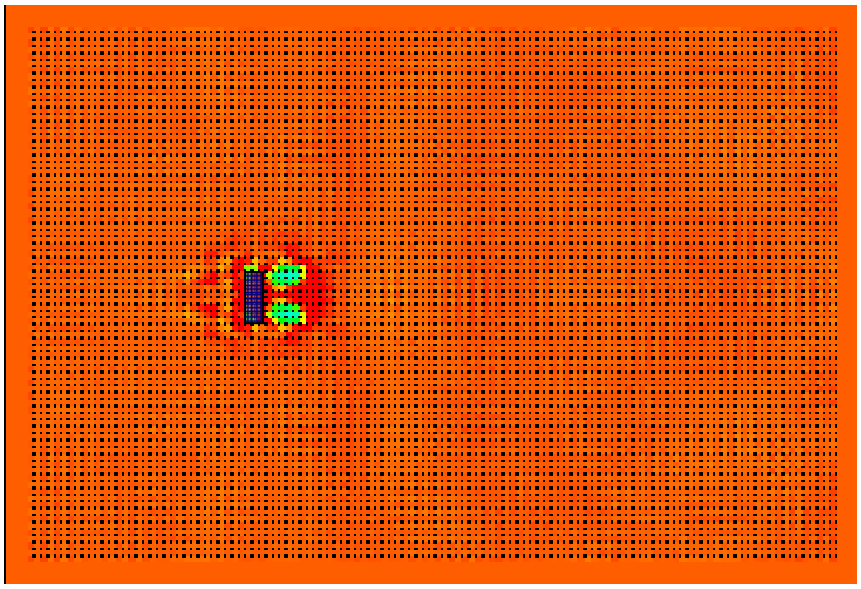


Figure 6.7: Flat plate density field, 88s

The density field in Figure 6.7 corresponds to a point in time 88 seconds after the start of the simulation. Here two plumes of cavitation are clearly forming just aft of the upper and lower corners. Hints of cavities are also visible at the leading corners though the resolution of this simulation is not high enough to fully capture their geometry. (This simulation uses just over 11,000 nodes.) Cavity density is approximately 447 kg/m^3 on average (the reference density of liquid water is 999.5 kg/m^3). Some noise in the density field is perceptible, particularly ahead of the plate, where pressure waves travel and interfere throughout the flow field. Suppression of this noise can be achieved by applying a low pass filter over the density field. This was tried, and while it does suppress the noise, it also smoothes the data too much removing important higher frequency features in the flow and resulting in too much numerical diffusion.

Approximately 10 seconds later in the simulation, the cavity densities have reduced further to approximately 1 kg/m^3 on average. As shown in Figure 6.8, these cavities are well defined with little smoothing across the interface boundary.

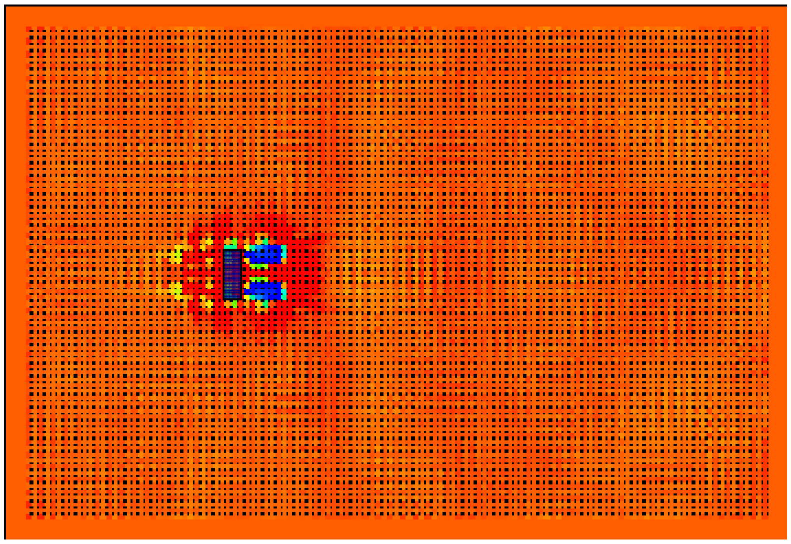


Figure 6.8: Flat plate density field, 97s

An interesting feature of this flow pattern is the pressure. The pressure within the cavities is low as expected, there exists a high pressure in the wake between the cavities. Figure 6.9 shows the pressure field corresponding to 88 seconds into the simulation. The color scale has been adjusted to highlight the high pressure regions. A high pressure region, with pressure greater than the reference pressure, is clearly visible between the cavities as indicated by the white arrow.

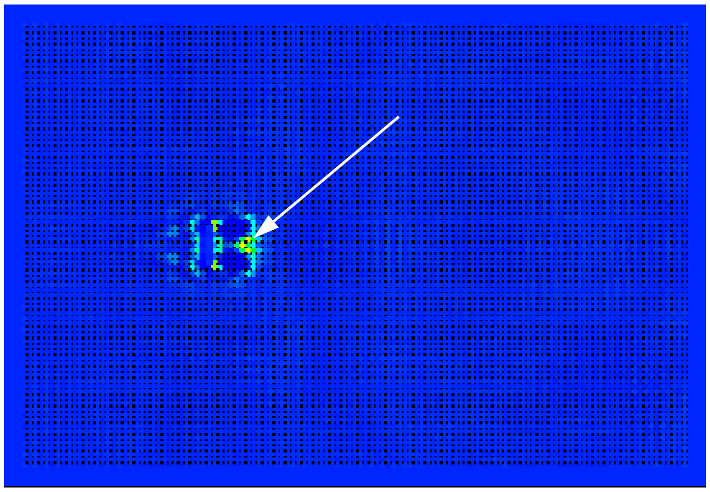


Figure 6.9: Pressure field at 88s

This high pressure region can be explained by examining the velocity field shown in Figure 6.10. Colors in Figure 6.10 indicated magnitude of fluid velocity at each node with warmer colors indicating higher speed. As expected the fluid velocity increases as the fluid flows around the corners of the plate. As the liquid flows around the vapor cavities they appear to come to a stagnation point where the flow bifurcates with flow traveling aft along the centerline of the plate as well as forward towards the back of the plate.

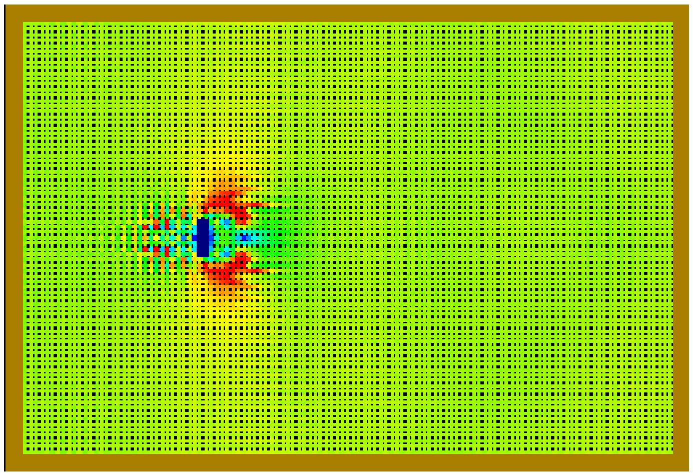


Figure 6.10: Velocity field

Figure 6.11 shows a zoomed in view of this situation from a similar simulation. In this figure the color scale again represents velocity magnitude and the white lines indicate velocity direction.

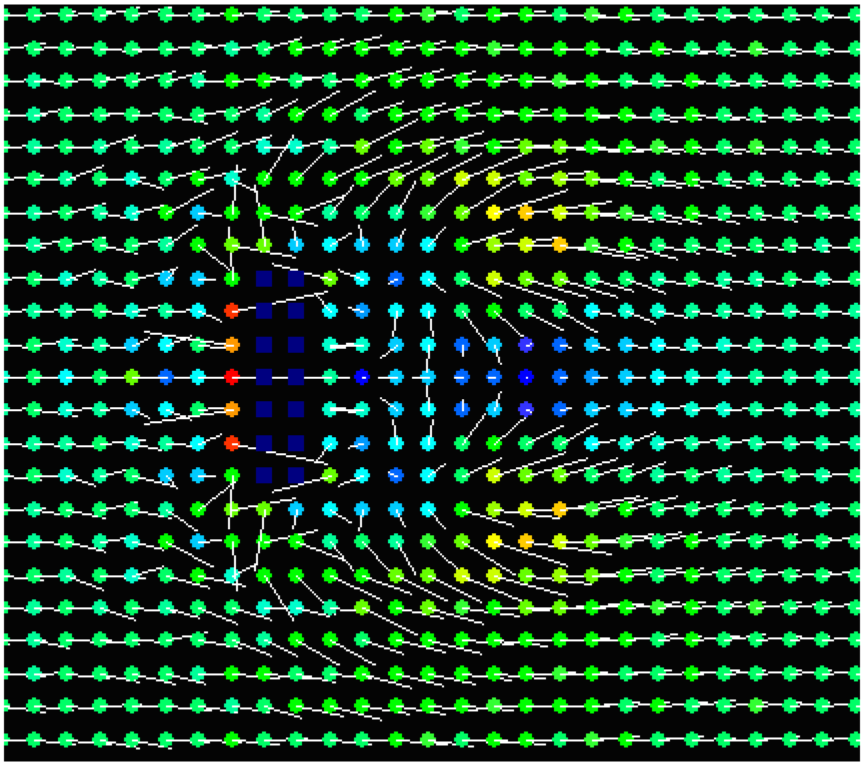


Figure 6.11: Velocity vector field

Approximately, 7 node spaces to the right of the plate exists a point where the velocity is zero. This point coincides with the region of high pressure. To the left of the this node, the velocity vector is towards the left and on the right side of this point the velocity vector is towards the right. This pattern is persistent over time. Flow within the cavities is generally directed outward in all directions while being swept back on the outer edges and directed forward along the centerline of the plate.

There also exists another point where the velocity is zero. This point is two node spaces to the right of the plate. However, unlike the stagnation point further to the right, this point is transient and seems due more to a pressure wave rebounding off the back side of the plate.

Wedge Model

The instability in the cavity closure area exhibited in the cavitating cylinder proved somewhat frustrating. Initial attempts to quell instability involved filtering the results to numerically smooth out the instability. This approach was successful in that smoothing was achieved, however, the cost was unacceptable. Smoothing had the adverse effect of removing the flow features of interest. In fact, too much smoothing actually caused the cavity to diffuse away over time. This, of course, was unacceptable and was thus abandoned.

The real solution was to revise the numerical integration scheme used to integrate the governing equations over time. My original implementation used a standard 4th order Runge-Kutta algorithm on the explicit governing equations. A revised algorithm using an implicit integration scheme for the continuity equation along with the original explicit scheme for the momentum equation proved successful. At this point, I chose to implement the revised integration scheme for a different problem, namely, that of supercavitating flow past a wedge. Figure 6.12 illustrates the problem.

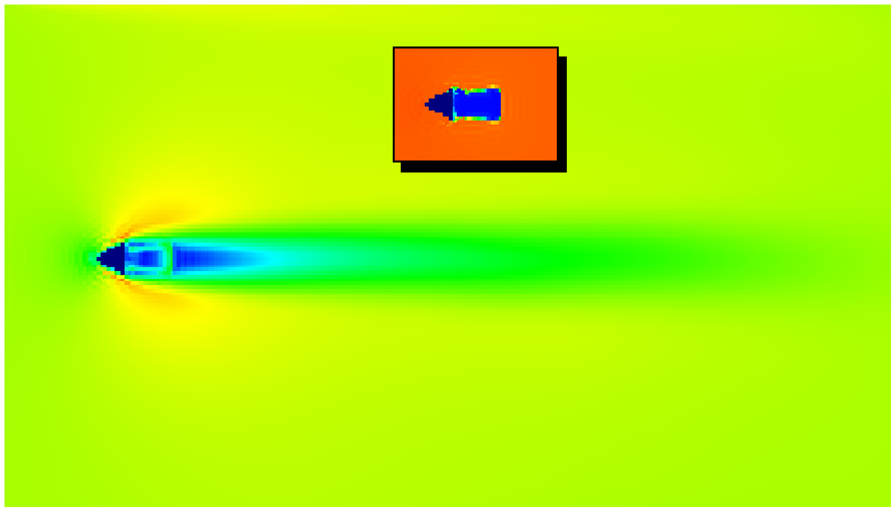


Figure 6.12: Supercavitating wedge

The primary illustration is that of the velocity field where the free stream velocity flows from left to right. The inset picture shows the corresponding density field with a fully developed cavity behind the wedge. The wedge has a unit chord length with unit width

Using $\gamma = 31$ instead of 7 as discussed earlier proved effective in these wedge tests. As shown in the figures throughout this section, the cavities are very well defined with little smoothing over the vapor-liquid interface. 1000:1 density ratios have also been achieved.

Cavity Length

I chose this wedge problem because I wanted to compare certain metrics with existing theoretical results. Reference [34] details a theory for modeling the fully developed cavitating flow around a wedge. This theory is based on potential flow theory and provides a means of estimating the length of the steady cavity behind the wedge. Using equations given in [34] I estimated the cavity length behind the unit wedge. The results are shown by the blue line labeled *Theory* in Figure 6.13. This theory predicts the cavity length will increase to infinity as the cavitation number approaches zero. (Cavity lengths shown in Figure 6.13 are non-dimensionalized in terms of wedge chord length.)

For comparison, I performed several numerical tests at three different cavitation numbers and estimated the cavity lengths resulting from each test. These results are also plotted in Figure 6.13 as represented by the red circles. In general, the numerical results agree very well with the theory particularly at the lower two cavitation numbers.

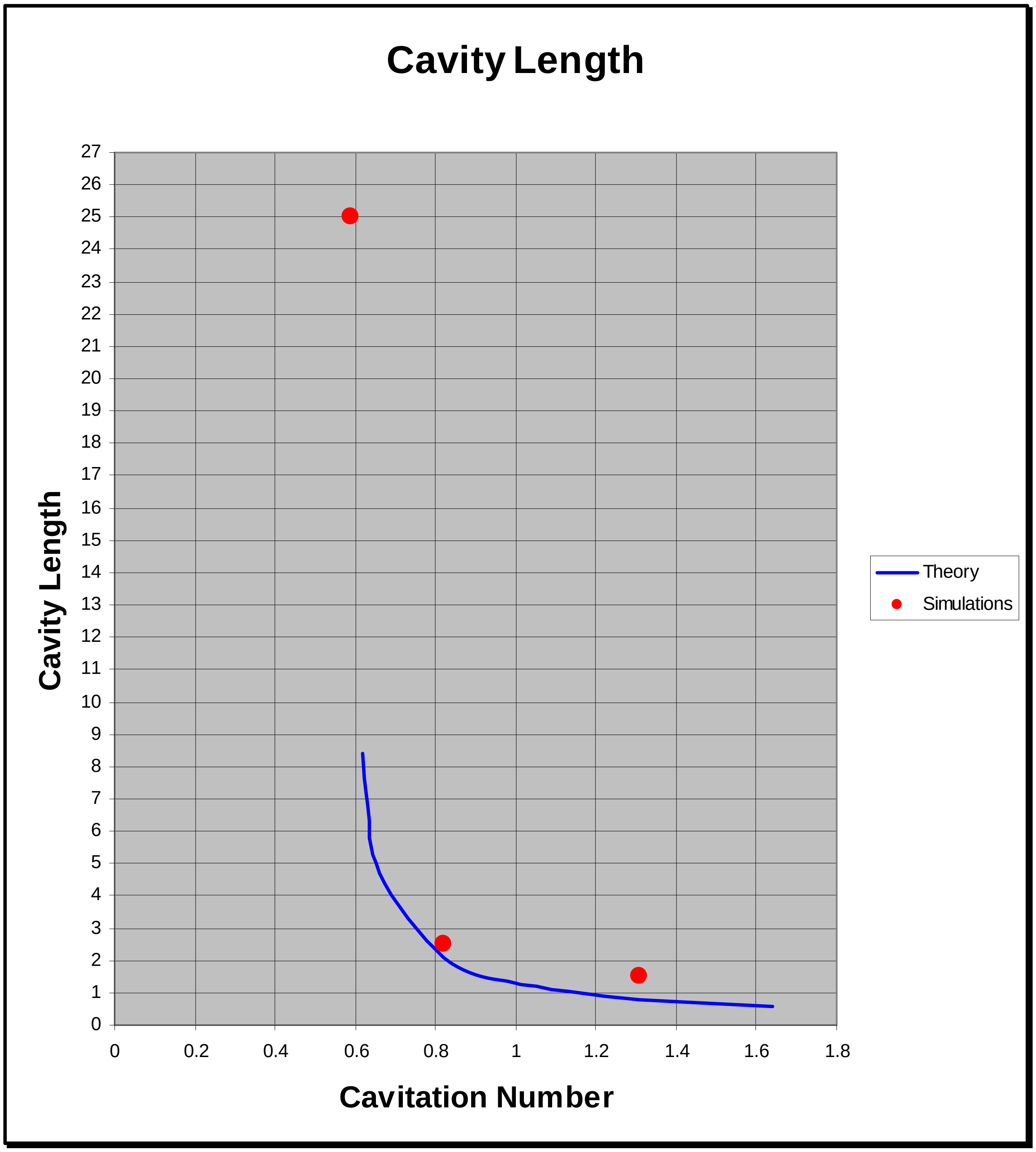


Figure 6.13: Cavity length comparison

Some of the differences in results can be explained by the fact that in the numerical tests, the cavity was not steady and its length varied depending on the degree to which bubbles or cavities where pinched off in the closure region.

Cavity Pinch

The revised integrator proved quite successful in stabilizing the method enough to enable long simulation run times capturing the pinch phenomenon without crashing. Figures 7.14 through 7.16 show some typical results for Reynold's number equal to 1000 and a cavitation number of 0.59. This sequence of images from the same simulation show the evolution of a large segment of a long cavity being pinched off.

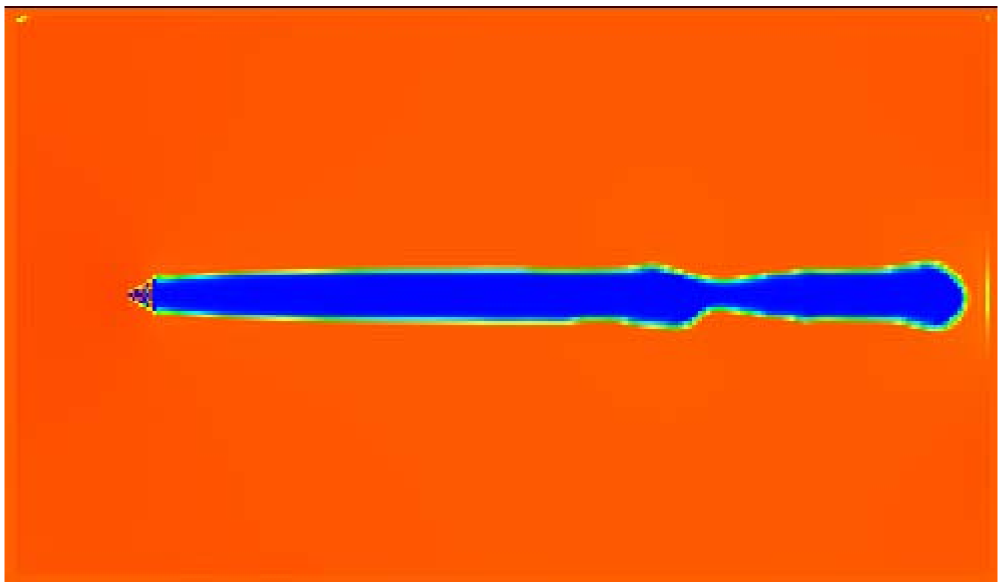


Figure 6.14: Cavity at 107.5 seconds

Figure 6.14 shows the cavity at 107.5 seconds into the simulation. At approximately 20 chord lengths aft of the wedge, the cavity begins to neck down. At 112s the cavity has nearly been pinched off and only a sliver remains connecting the attached cavity and the detaching cavity.

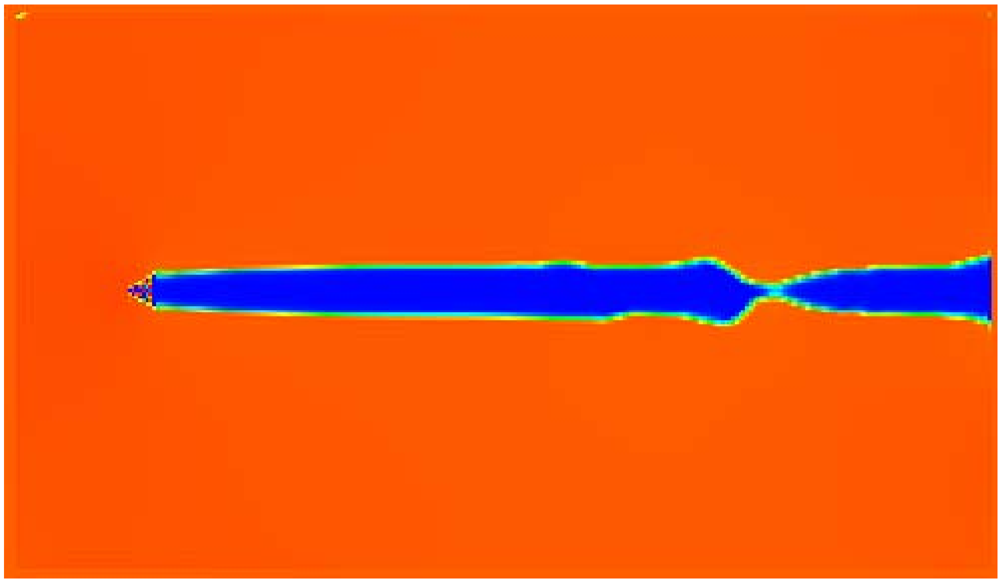


Figure 6.15: Cavity at 112 seconds

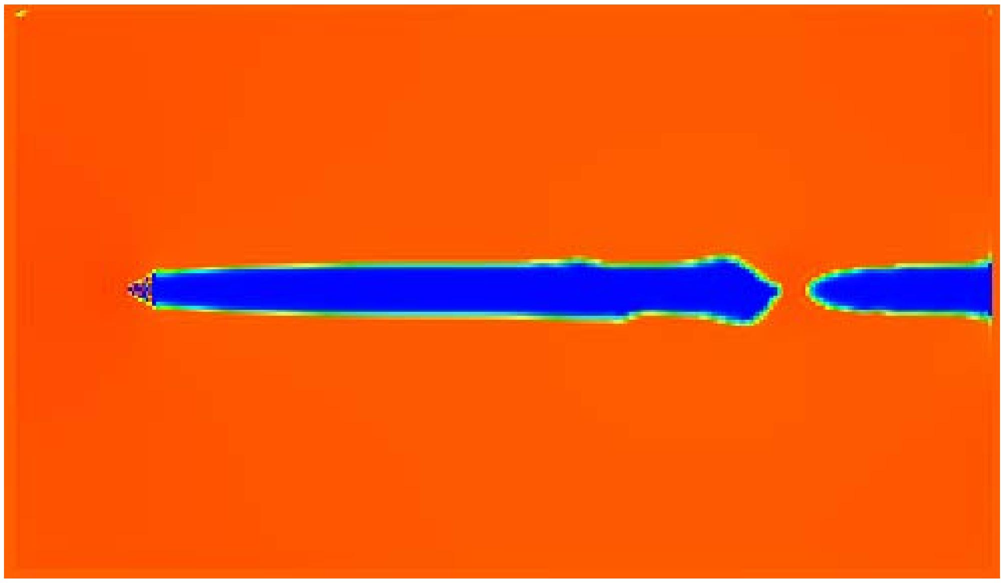


Figure 6.16: Cavity at 113.5 seconds

At 113.5 seconds the split is complete and the detached cavity is advected downstream. The cycle then repeats.

Tests at higher cavitation numbers exhibited similar behavior. For example, the same wedge test run at a cavitation number of 1 resulted in a cavity length of roughly 2 chord lengths, but this length was not steady. The cavity itself did not assume a steady shape, nor did the closure region.

Further, parts of the cavity were repeatedly pinched off at the trailing edge of the cavity. Figure 6.17 illustrates a sequence of successive frames show the cavity being pinched off.

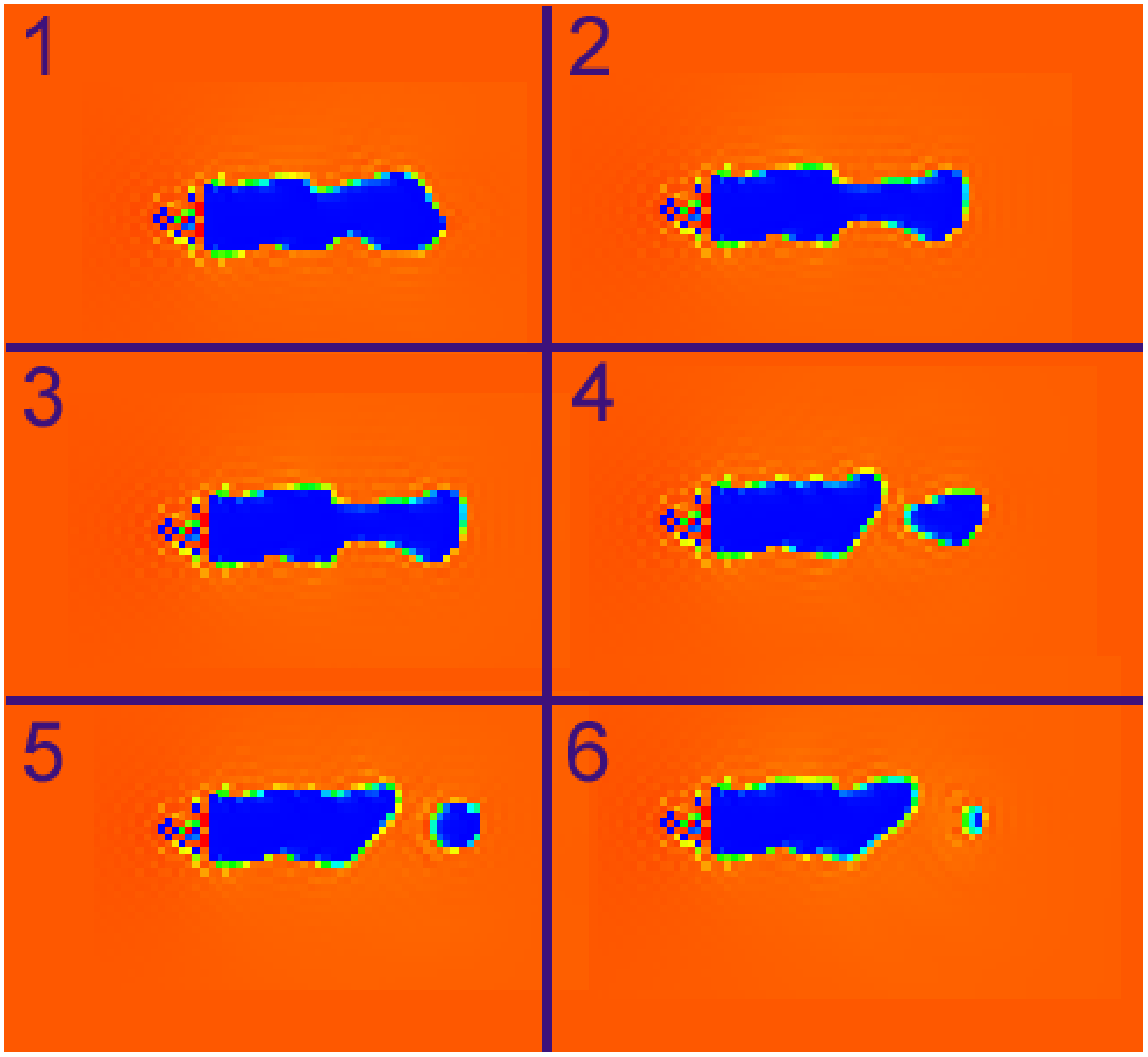


Figure 6.17: Cavity pinch sequence

As in all other tests, a large pressure pulse occurs at the point the pinched off cavity implodes. Figure 6.18 shows the pressure field immediately after the trailing bubble shown in frame 6 above implodes.

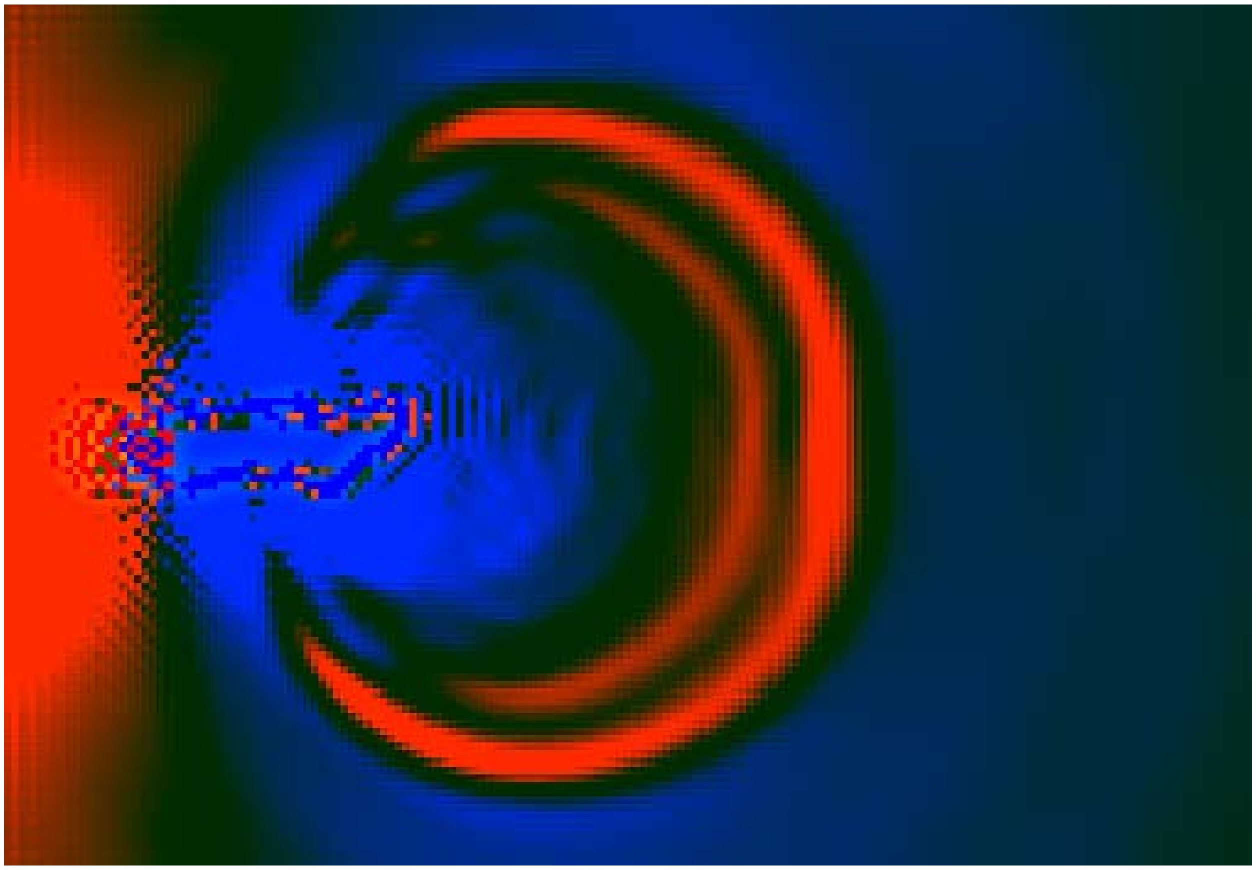


Figure 6.18: Pressure field at instant after bubble implosion

The large red concentric bands represent high pressure waves emanating from the point of bubble collapse and propagating at the artificial sound speed. The colors shown in Figure 6.18 have been adjusted to highlight the pressure waves.

Another characteristic of the pressure field, consistent in all tests, was that the pressure within the cavity was not constant but varied throughout the cavity. This is contrary to theory where it is assumed the pressure is uniform throughout the cavity. Another characteristic of the pinch off phenomenon is that it seems to occur when the velocity within the cavity increases inward from the outer edges spreading inward crossing the cavity width. Cavity pinch off seems to coincide with the moment this velocity transition completely traverses the cavity width. Figure 6.19 shows the velocity field for the lower cavitation number tests.

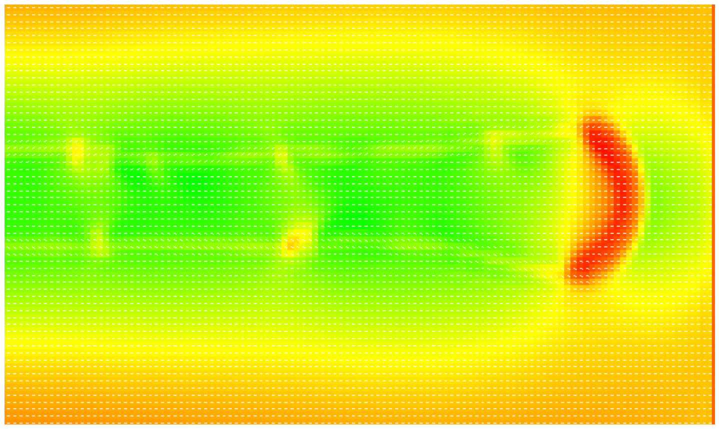


Figure 6.19: Band of high velocity

Warmer colors in this image represent high velocity. The prominent red band toward the right represents the velocity band of high velocity that has just completed formation across the width of the cavity. The cavity then breaks off just aft of the velocity band as illustrated earlier in Figure 6.16. The high velocity band isn't always vertically symmetric. In some instances within the same simulation the band is skewed towards the upper right or sometimes toward the bottom left. While, the pinch off phenomenon is recurrent within a simulation I have not taken measurements that would allow one to determine if the pinch off phenomenon is periodic.

The higher cavitation number tests also showed similar behavior as just described, however, the variability in pressure within the cavity was greater and the velocity bands seemed to develop at a higher frequency. It appears that the lower cavitation number tests with the relatively long cavity tends to stretch out the pressure and velocity fields within the cavity.

Towards the Future

The research presented herein focused on the development of the Distributed Points Method and its application to a variety of flows including cavitating flow. Many successes were demonstrated and discussed along with areas that should be the subject of further development. This final section briefly discusses several areas where I believe fruitful research should continue.

The single phase method was very successful in dealing with internal flow and Multifluid problems using both Eulerian and Lagrangian formulations. One specific area not covered in this research is that of free surface flow. I believe the Lagrangian formulation can be used with great success for treating free surface flow problems. For the most part, there should be few required modifications to the current method in order to handle free surface problems. The one obvious area requiring some treatment is how to handle the free surface boundary condition. Initial studies should focus on whether or not the DPM method is capable of handling the free surface boundary condition, namely the zero gauge pressure condition, naturally in a manner similar to SPH. However, that may not be possible in which case a two fluid model where one fluid represents air and the other water might be employed. This approach may be very useful in analyzing the sort of bubbly flow characteristic of ship bow waves where air is entrained and mixed with water. In this latter approach there are few required changes to the current method. A problem can be set up in a manner similar to the multifluid problem already discussed in Chapter 3.

Another area of the current method that can benefit from further research is extension of the method to 3D. Theoretically all that is required to do so is add another momentum equation to handle the z-coordinate and add appropriate z-terms to the other governing equations. Additionally, one would have to modify the weight function to approximate the 3D Gaussian function (or a linear model) and revise the shape functions used in the least squares approach to incorporate the z-coordinate. In practice, there will exist several numerical challenges, namely, optimizing the code for the added burden of a substantial number of computation nodes, revising the nearest neighbor search algorithm, and developing efficient least squares function fitting algorithms.

The issue of turbulence modeling discussed in Chapter 4 deserves further research. It would be interesting to see if the current method is capable of naturally capturing turbulence without the addition of specific turbulence models. If the method cannot, then it would be desirable to adapt existing turbulence models to the method. This seems like a relatively straightforward proposition. The first problem I'd tackle as part of this investigation would be high Reynold's number flow between two infinite flat plates. It would be interesting to see if the parabolic velocity profile discussed in Chapter 3 for this problem flattens out approaching a turbulent velocity profile as Reynold's number is increased.

It would seem that the above mentioned modifications would also benefit further research into the cavitation closure problem. I feel that further research should involve higher resolution tests aimed at better capturing the velocity field within the cavity in an effort to better understand how the velocity field relates to the pinch off phenomenon. Frequency analysis of the pinch off phenomenon should also be conducted in order to understand its periodic or non-periodic nature.

These additional studies could conceivably be captured using the current method, however, it would be very worthwhile to first parallelize the current method so that it can be run on a powerful computer cluster instead of a scalar computer. This would reduce computation time tremendously allowing for much higher resolution simulations, which I believe are required in order to fully capture the dynamics of the velocity field within the cavity and in way of the closure area. Numerical stability in the closure area has already been recognized as an issue that requires continued attention and the use of smaller time steps or other, perhaps more computationally intensive, integration schemes would certainly benefit from a parallel implementation of the method.

[1] Cleary, and Ha, "Three Dimensional Modeling of High Pressure Die Casting," Second International Conference on CFD in the Minerals and Process Industries CSIRO, Melbourne, Australia, 6-8 December 1999.

[2] Cleary, Ha, Alguine, and Nguyen "Simulation of Die Filling in Gravity Die Casting Using SPH and MAGMAsoft," Second International Conference on CFD in the Minerals and Process Industries CSIRO, Melbourne, Australia, 6-8 December 1999.

[3] Cummins, and Rudman, "An SPH Projection Method," Journal of Computational Physics 152, 584-607 (1999).

[4] Golanski, and Woolfson, "A smoothed particle hydrodynamics simulation of the collapse of the interstellar medium," Mon. Not. R. Astron. Soc. 320, 1-11 (2001).

[5] Katz, and O'Hern, "Cavitation in Large Scale Shear Flows," Journal of Fluids Engineering, Sept. 1986, Vol. 108/373.

[6] Li, and Liu, "Meshfree and Particle Methods and Their Applications," Applied Mechanics Review, 27 June 2001.

[7] Martin, Pearce, and Thomas, "An Owner's Guide to Smoothed Particle Hydrodynamics," J. Astro. Phys. 13 Oct. 1993.

[8] McCormick, and Miller, "Application of Smoothed Particle Hydrodynamics to Fluid Flow Involving Solid Boundaries," Technical Report No. TR 96-006, June 1996, Supported by: Space and Naval Warfare Systems Command.

[9] Monaghan, "On the Problem of Penetration in Particle Methods," Journal of Computational Physics 82, 1-15 (1989).

[10] Monaghan, "Shock Simulation by the Particle Method SPH," Journal of Computational Physics 52, 374-389 (1983).

[11] Monaghan, "Smoothed Particle Hydrodynamics," Annu. Rev. Astron. Astrophys, 1992, 30: 543-74.

[12] Monaghan, and Gingold, "Smoothed Particle Hydrodynamics: Theory and Application to non-spherical Stars," Mon. Not. R. Astr. Soc. (1977) 181, 375-389.

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

[14] Morris, Fox, and Zhu, "Modeling Low Reynolds Number Incompressible Flows Using SPH," Journal of Computational Physics 136, 214-226 (1997).

[15] Rasio, "Particle Methods in Astrophysical Fluid Dynamics," J. Astro. Phys. 18 Nov. 1999.

[16] Ritchie, and Thomas, "Multiphase Smoothed-Particle Hydrodynamics," Mon. Not. R. Astron. Soc, 21 April 2001.

[17] Sawley, Cleary, and Ha, "Modeling of Flow in Porous Media and Resin Transfer Moulding Using Smoothed Particle Hydrodynamics," Second International Conference on CFD in the Minerals and Process Industries CSIRO, Melbourne, Australia, 6-8 December 1999.

[18] Schlateter, Brian, "A Pedagogical Tool Using Smoothed Particle Hydrodynamics to Model Fluid Flow Past a System of Cylinders," Oregon State University Masters Thesis, June 15, 1999.

[19] Stein, and Max, "A Particle-Based Model for Water Simulation," SIGGRAPH '98, Orlando, FL, July 19-24, 1998.

[20] Vertanen, Keith, "A Parallel Implementation of a Fluid Flow Simulation using Smoothed Particle Hydrodynamics," Oregon State University Masters Thesis, June 24, 1999.

[21] Watanabe, Seki, Higashi, Yokota, Tsujimoto, "Modeling of 2-D Leakage Jet Cavitation as a Basic Study of Tip Leakage Vortex Cavitation," Transactions of the ASME, Vol. 123, March 2001.

[22] Watkins, Bhattal, Francis, Turner, and Whitworth, "A new prescription for viscosity in Smoothed Particle Hydrodynamics," Astron. Astrophys. Suppl. Series 119, 177-187 (1996).

[23] Koshizuka, and Oka, "Moving Particle Semi-Implicit Method: Fully Lagrangian Analysis of Incompressible Flows," ECCOMAS 2000, 11-14 Sept. 2000.

[24] Liu, G. R., and Liu, M. B., Smoothed Particle Hydrodynamics, a meshfree particle method, World Scientific Publishing Co. Pte. Ltd., 2003.

[25] Lo, Edmond Y. M., Shao, Songdong, "Simulation of near-shore solitary wave mechanics by an incompressible SPH method," Applied Ocean Research 24, 2002.

[26] Takeda, Miyama, and Sekiya, "Numerical Simulation of Viscous Flow by Smoothed Particle Hydrodynamics," Progress of Theoretical Physics, Vol. 92, No. 5, Nov. 1994.

[27] Muller, Charypar, Gross, "Particle-Based Fluid Simulation for Interactive Applications," Eurographics, SIGGRAPH Symposium on Computer Animations, 2003.

[28] Flebbe, Munzel, Herold, Riffert, and Ruder, "Smoothed Particle Hydrodynamics: Physical Viscosity and the Simulation of Accretion Disks," The Astrophysical Journal, 431:754-760, Aug. 1994.

[29] Cleary and Monaghan, "Boundary interactions and transition to turbulence for standard CFD problems using SPH"

[30] Marri and White, "Smoothed particle hydrodynamics for galaxy-formation simulations: improved treatments of multiphase gas, of star formation and of supernovae feedback," Mon. Not. R. Astron. Soc. 345, 2003.

[31] Colagrossi and Landrini, "Numerical simulation of interfacial flows by smoothed particle hydrodynamics," Journal of Computational Physics 191, 2003.

[32] Brennen, Christopher E., Cavitation and Bubble Dynamics, Oxford University Press, 1995.

[33] Coutier-Delgosha, and Astolfi, "Numerical Prediction of the Cavitating Flow on a Two Dimensional Symmetrical Hydrofoil with a Single Fluid Model," Workshop on physical models and CFD tools for computation of cavitating flows, Fifth International Symposium on Cavitation, Osaka, Japan, November 2003.

[34] Newman, J. N., Marine Hydrodynamics, The MIT Press, 1977.

[35] Reboud, Pouffary, Coutier-Delgosha, and Patella, "Numerical Simulation of Unsteady Cavitating Flows: Some Applications and Open Problems," Workshop on physical models and CFD tools for computation of cavitating flows, Fifth International Symposium on Cavitation, Osaka, Japan, November 2003.

[36] Katz, and O'Hern, "Cavitation in Large Scale Shear Flows," Journal of Fluids Engineering, September, 1986, Vol. 108.

[37] Belahadji, and Michel, "Numerical and Experimental Study of Cavitating Vortices in Turbulent Wake.

[38] Watanabe, Seki, Higashi, Yokota, Tsujimoto, "Modeling of 2D Leakage Jet Cavitation as a Basic Study of Tip Leakage Vortex Cavitation," Transactions of the ASME, Vol. 123, March 2001.

[39] Landrini, Colagrossi, Tulin, "A Novel SPH Formulation for 2-Phase Flows"

[40] Vachem, Almstedt, "Methods for multiphase computation fluid dynamics," Chemical Engineering Journal 96, 2003.

[41] Kunz, Lindau, Billet, Stinebring, "Multiphase CFD modeling of Developed and Supercavitating Flows"

[42] Zhang and Chen, "Lattice Boltzmann method for simulations of liquid-vapor thermal flows," Physical Review 67, 2003.

[43] Chaniotis, Frouzakis, Lee, Tomboulides, Poulikakos, Boulouchos, "Remeshed smoothed particle hydrodynamics for the simulation of laminar chemically reactive flows," Journal of Computational Physics 191, 2003.

[44] Jeffery and Austin, "A new analytic equation of state for liquid water," Journal of Chemical Physics, Vol. 110, No. 1, 1998.

[45] Tryggvason, Bunner, Ebrat, and Tauber, "Computations of Multiphase Flows by a Finite Difference/Front Tracking Method. I. Multi-Fluid Flows."

[46] Hong, Xin, and Huayong, "Numerical Simulation of Cavitating Flow in Hydraulic Conical Valve."

[47] Chen and Weng, "Analysis of Flow Past a Two-Dimensional Supercavitating Hydrofoil Using a Simple Closure Model," Proc. Natl. Sci Counc. ROC(A), Vol. 23, No. 3, 1999.

[48] Berntsen, Kjeldsen, Arndt, "Numerical Modeling of Sheet and Tip Vortex Cavitation with FLUENT 5," CAV 2001.

[49] Frobenius, Schilling, Friedrichs, and Kosyna, "Numerical and experimental investigations of the cavitating flow in a centrifugal pump impeller," Proc. Of FEDSM '02, ASME Fluids Engineering Division Summer Meeting, Montreal, Quebec, Canada, July 2002.

[50] Qin, Yu, Zhang, and Lai, "Direct Calculations of Cavitating Flows by the Space-Time CE/SE Method,"

[51] Fluent Users Guide, Fluent Inc. 2003.

[52] Senocak and Shyy, "Evaluation of Cavitation Models for Navier-Stokes Computations," FEDSM2002-31011.

[53] Farrell, Kevin, "Eulerian/Lagrangian Analysis for the Prediction of Cavitation Inception," Transactions of the ASME, Vol. 125, Jan. 2003.

[54] Coutier-Delgosha, Fortes-Patella, and Rebound, "Evaluation of the Turbulence Model Influence on the Numerical Simulations of Unsteady Cavitation," Transactions of the ASME, Vol. 125, Jan. 2003.

[55] Owen, Villumsen, Shapiro, and Martel, "Adaptive Smoothed Particle Hydrodynamics: Methodology. II.," The Astrophysical Journal Supplement Series, June 1998.

[56] Schick, "Adaptivity for Particle Methods in Fluid Dynamics," Thesis, University of Kaiserslautern, Department of Mathematics, 2003.

[57] Castano-Moraga, Rodriguez-Florido, Alvarez, Westin, and Ruiz-Alzola, "Anisotropic Interpolation of DT-MRI," MICCAI 2004.

[58] Geusebrock, Smeulders, van de Weijer, "Fast Anisotropic Gauss Filtering," IEEE Transactions on Image Processing, Vol. 12, No. 8, Aug 2003.

[59] Foi, Katkovnik, Egiazarian, Astola, "A novel anisotropic local polynomial estimator based on directional multiscale optimizations."

[60] Lo, and Shao, "Simulation of near-shore solitary wave mechanics by an incompressible SPH method," Applied Ocean Research 24, 2002.

[61] Welton, and Pope, "PDF Model Calculations of Compressible Turbulent Flows Using Smoothed Particle Hydrodynamics," Journal of Computational Physics, 134, 1997.

[62] Violeau, Piccon, Chabard, "Two Attempts of Turbulence Modeling in Smoothed Particle Hydrodynamics," FMTM 2001.

[63] Chahine, "Nuclei Effects on Cavitation Inception and Noise," 25th Symposium on Naval Hydrodynamics, August 2004.

[64] Hsiao and Chahine, "Numerical Study of Cavitation Inception due to Vortex/Vortex Interaction in a Ducted Propulsor," 25th Symposium on Naval Hydrodynamics, August 2004.

[65] Hsiao and Chahine, "Prediction of Vortex Cavitation Inception Using Coupled Spherical and Non-Spherical Models and UnRANS Computations," 24th Symposium on Naval Hydrodynamics, July 2002.

[66] Liu and Brenned, "Cavitation Nuclei Population and Event Rates," Transactions of the ASME, 1998.

[67] Liu and Brennen, "The Relation Between the Nuclei Population and the Cavitation Event Rate for Cavitation on a Schiebe Body," FED-Vol. 190, Cavitation and Gas-Liquid Flow in Fluid Machinery and Devices, ASME 1994.

[68] Bashir, Bilal, and Khan, "Numerical Simulation of Cavitation Inception for Axisymmetric Headforms," 2nd International Bhurban Conference on Applied Sciences and Technology, June 2003.

[69] Maitre and Pellone, "Numerical Modeling of Unsteady Partial Cavities Behind a Backward Facing Step," 4th International Symposium on Cavitation, June 2001.

[70] Ventikos and Tzabiras, "A numerical method for the simulation of steady and unsteady cavitating flows," Computers and Fluids 29, 2000.

[71] Koumoutsakos, Petros, "Multiscale Flow Simulations Using Particles," Annual Review of Fluid Mechanics, 2005, 37:457-87.

[72] Press, Teukolsky, Vetterline, and Flannery, Numerical Recipes in C, 2nd Edition, Cambridge University Press, 1992.

[73] Rouse, Hunter, Elementary Mechanics of Fluids, Dover Publications, Inc, New York, 1946.

[74] Minero, R., "Mesh Free Methods for Fluid Dynamics Problems," CASA, Dec. 17, 2003.

[75] Bruneau and Saad, "The 2D lid-driven cavity problem revisited," Computers & Fluids 35 (2006), pp 326-348.

[76] Blevins, Robert D., Applied Fluid Dynamics Handbook, Krieger Publishing Company, 2003.

[77] Rahman, Karim, and Alim, "Numerical Investigation of Unsteady Flow Past a Circular Cylinder Using 2-D Finite Volume Method," Journal of Naval Architecture and Marine Engineering, June, 2007.

Appendix

SELECTED SOURCE CODE

Included in this appendix are selected code samples from my implementation of DPM. These samples were selected to show how key aspects of the method was implemented. Two versions of DPM were implemented as part of this research effort. The first version was written in Object Pascal while the second was written in C/C++. Borland developer tools were used for both versions.

Borland Object Pascal as implement in Borland's Delphi developer tool was selected for the first version due to its ease of developing Windows GUI applications. This version proved quite useful and most of the simulations discussed in this dissertation were made using this first version. The second version was implemented in C/C++ using Borland's C++ Builder development tool. This switch to C++ was made in an effort to optimize the code for performance. The DPM methodologies used in both versions were identical. Only algorithmic changes were made in the C++ version in order to streamline the code for performance reasons.

TakeStep

This function is called once per time step during a simulation. It implements an adaptive time stepping scheme whereby a single step, from to, is taken at the full time step size and then two half steps are taken, from to, and the results compared. Depending on the error, the time step is adjusted upward or downward.

```
void    TWedgeModel::TakeStep(bool firstStep)
{
        int                     r,c;
        TParticle*              pptr;
        double                  origStepSize = StepSize;
        double                  m = 1;

        if(CAVITATION) {
                Max_dt = 0;
                Max_ceff = 0;

                if(SimTime > CAV_RAMP_TIME)
                {
                        if(SimTime > CAV_RAMP_TIME)
                                CavParam = CAVPARAM;
                }
        }

        // Take whole step:
        SimTime += StepSize;
        ComputeAllFields_RK4(firstStep, ADAPTIVE_TIME_STEP);

        // If using adaptive time stepping, take two half steps:
        if(ADAPTIVE_TIME_STEP)
        {
                StepSize = StepSize / 2.0;
                ComputeAllFields_RK4(firstStep, false);
                ComputeAllFields_RK4(firstStep, false);
                ComputeErrors();
                MaxError = max(MaxVelocityError,
                           max(MaxPressureError, MaxDensityError));

                if(MaxError != 0)
                {
                        if(MaxError > TOLERANCE)
                        {
                                m = 0.97;//TOLERANCE / MaxError;
                                StepSize = origStepSize *
                                pow(TOLERANCE / MaxError, 0.2) * m;

                        } else
                                StepSize = origStepSize *
                                    pow(TOLERANCE / MaxError, 0.2);
                }

                StepSize = min(StepSize, Max_dt);

                if(MaxError > TOLERANCE)
                {
                        CopyData_tm_to_t0(); // FORCES A REPEATED STEP
                        SimTime -= origStepSize;
                } else {
                        if(SimTime <= RAMP_TIME)
                        {
                                WedgeSpeed -= RAMP_STEP;
                                if(WedgeSpeed < 0)
                                        WedgeSpeed = 0;
                        } else {
                                WedgeSpeed = 0;
                        }
                }
        } else {
                if(SimTime <= RAMP_TIME)
                {
                        WedgeSpeed -= RAMP_STEP;
                        if(WedgeSpeed < 0)
                                WedgeSpeed = 0;
                } else {
                        WedgeSpeed = 0;
                }
        }
}
```

ComputeAllFields_RK4

This function is called from *TakeStep* and implements a fourth order Runge-Kutta integration scheme. The momentum and continuity equations are solved here and new pressures are also computed.

```
void    TWedgeModel::ComputeAllFields_RK4(bool firstStep, bool saveData1)
{
        double                  ku[4], kv[4], kr[4];
        int                     r,c;
        TParticle*              pptr;
        double                  vs = 2*PLOT_SCALE_V_MAX;
        double                  dt = StepSize;

        // k1 step: Density & Pressure
        GetGradients(true, false, true);
        for(r=0; r<NUM_ROWS; r++)
        {
                for(c=0; c<NUM_COLS; c++)
                {
                        memcpy(&(Particle[r][c].Data_tm),
                    &(Particle[r][c].Data_t0), sizeof(TFieldData));

                        // get pointer to current particle
                        pptr = &(Particle[r][c]);

                        // integrate continuity equation
                        IntegrateContinuityEquation(pptr);
                        kr[0] = pptr->Data_t0.dRdt;
                        pptr->Data_t0.Density = pptr->Data_tm.Density +
                                                   dt * kr[0] / 2;

                        // compute mixture viscosity
                        pptr->Data_t0.mu = (pptr->Data_t0.Density /
                                          RHO_LIQUID) * MU_LIQUID +
                                        (1 - (pptr->Data_t0.Density
                                         / RHO_LIQUID)) * MU_VAPOR;

                        // compute new pressure pressure
                        ComputePressures(pptr);

                } // c
        } // r

        // k1 step: Velocity
        GetGradients(true, true, false);
```

```
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                // integrate momentum equations
                // only if fluid particle
                if((pptr->ParticleType == tptFLUID) ||
                 (c >= NUM_COLS - 3))
                {
                        IntegrateMomentumEquations(pptr);
                        ku[0] = pptr->Data_t0.dUdt;
                        kv[0] = pptr->Data_t0.dVdt;

                        pptr->Data_t0.Velocity.x =
                              pptr->Data_tm.Velocity.x +
                              dt * ku[0] / 2;
                        pptr->Data_t0.Velocity.y =
                              pptr->Data_tm.Velocity.y +
                              dt * kv[0] / 2;
                }
        } // c
} // r

// k2 step: Density & Pressure
GetGradients(true, false, true);
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                 // integrate continuity equation
                 IntegrateContinuityEquation(pptr);
                 kr[1] = pptr->Data_t0.dRdt;
                 pptr->Data_t0.Density = pptr->Data_tm.Density +
                                         dt * kr[1] / 2;

                 // compute mixture viscosity
                 pptr->Data_t0.mu = (pptr->Data_t0.Density /
                                    RHO_LIQUID) * MU_LIQUID +
                                    (1-(pptr->Data_t0.Density /
                                    RHO_LIQUID)) * MU_VAPOR;

                 // compute new pressure pressure
                 ComputePressures(pptr);

        } // c
} // r

// k2 step: Velocity
```

```
GetGradients(true, true, false);
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                // integrate momentum equations
                // only if fluid particle
                if((pptr->ParticleType == tptFLUID) ||
                   (c >= NUM_COLS - 3))
                {
                        IntegrateMomentumEquations(pptr);
                        ku[1] = pptr->Data_t0.dUdt;
                        kv[1] = pptr->Data_t0.dVdt;

                        pptr->Data_t0.Velocity.x =
                              pptr->Data_tm.Velocity.x +
                              dt * ku[1] / 2;
                        pptr->Data_t0.Velocity.y =
                              pptr->Data_tm.Velocity.y +
                              dt * kv[1] / 2;
                }
        } // c
} // r


// k3 step: Density & Pressure
GetGradients(true, false, true);
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                 // integrate continuity equation
                 IntegrateContinuityEquation(pptr);
                 kr[2] = pptr->Data_t0.dRdt;
                 pptr->Data_t0.Density = pptr->Data_tm.Density +
                                         dt * kr[2];

                 // compute mixture viscosity
                 pptr->Data_t0.mu = (pptr->Data_t0.Density /
                                    RHO_LIQUID) * MU_LIQUID +
                                    (1 - (pptr->Data_t0.Density /
                                    RHO_LIQUID)) * MU_VAPOR;

                 // compute new pressure pressure
                 ComputePressures(pptr);

        } // c
} // r
```

```
// k3 step: Velocity
GetGradients(true, true, false);
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                // integrate momentum equations
                // only if fluid particle
                if((pptr->ParticleType == tptFLUID) ||
                   (c >= NUM_COLS - 3))
                {
                        IntegrateMomentumEquations(pptr);
                        ku[2] = pptr->Data_t0.dUdt;
                        kv[2] = pptr->Data_t0.dVdt;

                        pptr->Data_t0.Velocity.x =
                              pptr->Data_tm.Velocity.x +
                              dt * ku[2];
                        pptr->Data_t0.Velocity.y =
                              pptr->Data_tm.Velocity.y +
                              dt * kv[2];
                }
        } // c
} // r

// k4 step: Density & Pressure
GetGradients(true, false, true);
for(r=0; r<NUM_ROWS; r++)
{
        for(c=0; c<NUM_COLS; c++)
        {
                // get pointer to current particle
                pptr = &(Particle[r][c]);

                 // integrate continuity equation
                 IntegrateContinuityEquation(pptr);
                 kr[3] = pptr->Data_t0.dRdt;
                 pptr->Data_t0.dRdt = (kr[0] + 2*kr[1] +
                                      2*kr[2] + kr[3]) / 6.0;
                 pptr->Data_t0.Density = pptr->Data_tm.Density +
                                      dt * pptr->Data_t0.dRdt;

                 // compute mixture viscosity
                 pptr->Data_t0.mu = (pptr->Data_t0.Density
                                   / RHO_LIQUID) * MU_LIQUID +
                                   (1 - (pptr->Data_t0.Density /
                                   RHO_LIQUID)) * MU_VAPOR;

                 // compute new pressure pressure
                 ComputePressures(pptr);
                } // c
        } // r

        // k4 step: Velocity
        GetGradients(true, true, false);
        for(r=0; r<NUM_ROWS; r++)
        {
                for(c=0; c<NUM_COLS; c++)
                {
                        // get pointer to current particle
                        pptr = &(Particle[r][c]);

                        // integrate momentum equations
                        // only if fluid particle
                        if((pptr->ParticleType == tptFLUID) ||
                            (c >= NUM_COLS - 3))
                        {
                                IntegrateMomentumEquations(pptr);
                                ku[3] = pptr->Data_t0.dUdt;
                                kv[3] = pptr->Data_t0.dVdt;

                                pptr->Data_t0.dUdt = (ku[0] + 2*ku[1]
                                            + 2*ku[2] + ku[3]) / 6.0;
                                pptr->Data_t0.dVdt = (kv[0] + 2*kv[1]
                                            + 2*kv[2] + kv[3]) / 6.0;

                                pptr->Data_t0.Velocity.x =
                                      pptr->Data_tm.Velocity.x +
                                      dt * pptr->Data_t0.dUdt;
                                pptr->Data_t0.Velocity.y =
                                      pptr->Data_tm.Velocity.y +
                                      dt * pptr->Data_t0.dVdt;
                        }

                        if(pptr->ParticleType == tptBODY)
                        {
                                pptr->Data_t0.Velocity.x = WedgeSpeed;
                        }

                        if(saveData1)
                        {
                                memcpy(&(Particle[r][c].Data_t1),
                                       &(Particle[r][c].Data_t0),
                                       sizeof(TFieldData));
                                memcpy(&(Particle[r][c].Data_t0),
                                       &(Particle[r][c].Data_tm),
                                       sizeof(TFieldData));
                        }
                } // c
        } // r
}
```

IntegrateContinuityEquation

This function computes the continuity equation as called from *ComputeAllFields_RK4*.

```
void    TWedgeModel::IntegrateContinuityEquation(TParticle* pptr)
{
        double  t1, t2, t3;

        try {
                t1 = ( pptr->Gradients.U.dx + pptr->Gradients.V.dy);
                t2 = pptr->Data_t0.Velocity.x * pptr->Gradients.R.dx;
                t3 = pptr->Data_t0.Velocity.y * pptr->Gradients.R.dy;
        } catch(...) {
                t1 = 0;
        }

        pptr->Data_t0.dRdt = -pptr->Data_t0.Density * t1 -(t2 + t3);
}
```

ComputePressures

This function computes new pressures based on updated densities as called from *ComputeAllFields_RK4*.

```
void    TWedgeModel::ComputePressures(TParticle* pptr)
{
        double  dpdr, tmp1, tmp2, tmp3, tmp4, tmp6, ceff;

        if(CAVITATION)
        {
                if( pptr->Data_t0.Density < (RHO_LIQUID/2.0))
                {
                           dpdr = pptr->Data_t0.c*pptr->Data_t0.c *
                                  (1-CavParam) * pow((RHO_LIQUID –
                                   pptr->Data_t0.Density)
                                   /RHO_LIQUID, GAMMA - 1);

                           ceff = sqrt(dpdr);

                           tmp1 = (ceff*ceff*RHO_LIQUID)/GAMMA;
                           tmp6 = (pptr->Data_t0.Density-
                                   RHO_LIQUID)/RHO_LIQUID;
                           tmp2 = tmp6/(fabs(tmp6)) *
                                   pow(fabs(tmp6), GAMMA);
                           tmp3 = pow((RHO_LIQUID –
                                  pptr->Data_t0.Density)/
                                  RHO_LIQUID, GAMMA);
                           tmp4 = (1-CavParam) * (1 - tmp3) + abs(tmp2);
                           pptr->Data_t0.Pressure = tmp1 * (tmp2 + tmp4);
                } else {
                           dpdr = pptr->Data_t0.c*pptr->Data_t0.c *
                                  ( pow(pptr->Data_t0.Density/
                                  RHO_LIQUID, GAMMA - 1) * CavParam);

                           ceff = sqrt(dpdr);
                           pptr->Data_t0.Pressure =
                                      ((ceff*ceff*RHO_LIQUID)/GAMMA) *
                                      ( ( pow((pptr->Data_t0.Density/
                                      RHO_LIQUID), GAMMA)) +
                                      (1-CavParam) * (1 –
                                      pow(pptr->Data_t0.Density/
                                      RHO_LIQUID, GAMMA)) );
                }
                Max_ceff = max(ceff, Max_ceff);
                Max_dt = max( (min(X_SPACING, Y_SPACING) /
                            Max_ceff) , Max_dt);
        } else {
                pptr->Data_t0.Pressure = pptr->Data_t0.c *
                            pptr->Data_t0.c * pptr->Data_t0.Density;

        }
}
```

IntegrateMomentumEquations

This function computes the momentum equations as called from *ComputeAllFields_RK4*.

```
void    TWedgeModel::IntegrateMomentumEquations(TParticle* pptr)
{
        int     signX = 1;
        int     signY = 1;

        // do x-momentum equation...
        pptr->Data_t0.dUdt = -pptr->Gradients.P.dx /
                             pptr->Data_t0.Density +
                             G_X - pptr->Data_t0.Velocity.x *
                             pptr->Gradients.U.dx –
                             pptr->Data_t0.Velocity.y *
                             pptr->Gradients.U.dy ;

        if(DO_VISCOSITY)
                pptr->Data_t0.dUdt += pptr->Data_t0.mu /
                                      pptr->Data_t0.Density *
                                      (pptr->Gradients.U.dxdx +
                                      pptr->Gradients.U.dydy);

        // do y-momentum equation...
        pptr->Data_t0.dVdt = -pptr->Gradients.P.dy /
                             pptr->Data_t0.Density +
                             G_Y - pptr->Data_t0.Velocity.x
```

```
                              * pptr->Gradients.V.dx +
                              - pptr->Data_t0.Velocity.y *
                              pptr->Gradients.V.dy ;

        if(DO_VISCOSITY)
                pptr->Data_t0.dVdt += pptr->Data_t0.mu /
                                      pptr->Data_t0.Density *
                                      (pptr->Gradients.V.dxdx +
                                      pptr->Gradients.V.dydy);
}
```

GetGradients

This function computes gradients of all field values.

```
void    TWedgeModel::GetGradients(bool doVelocity, bool doPressure, bool
doDensity)
{
        double          x[NUM_NEIGHBORS];
        double          y[NUM_NEIGHBORS];
        double          u[NUM_NEIGHBORS];
        double          v[NUM_NEIGHBORS];
        double          uxx[NUM_NEIGHBORS];
        double          uyy[NUM_NEIGHBORS];
        double          vxx[NUM_NEIGHBORS];
        double          vyy[NUM_NEIGHBORS];
        double          p[NUM_NEIGHBORS];
        double          d[NUM_NEIGHBORS];
        double          w[NUM_NEIGHBORS];
        double          sig[NUM_NEIGHBORS];
        double          s;
        double          fx;
        double          fy;
        double          fxx;
        double          fyy;
        double          px;
        double          py;
        int             r, c, i;
        TParticle*      fptr;
        TFieldData*     ifptr;

        for(r=0; r<NUM_ROWS; r++)
        {
                for(c=0; c<NUM_COLS; c++)
                {
                        px = Particle[r][c].Data_t0.Position.x;
                        py = Particle[r][c].Data_t0.Position.y;

                        Particle[r][c].Data_t0.AveV.x = 0;
                        Particle[r][c].Data_t0.AveV.y = 0;
                        Particle[r][c].Data_t0.AveP = 0;
```

```
Particle[r][c].Data_t0.AveR = 0;

ifptr = (TFieldData*) &(Particle[r][c].Data_t0);

for(i=0; i<NUM_NEIGHBORS; i++)
{
        fptr = (TParticle*)
               Particle[r][c].Neighbors[i];

        x[i] = (fptr->Data_t0.Position.x - px) *
               Particle[r][c].NMult[i].x;
        y[i] = (fptr->Data_t0.Position.y - py) *
               Particle[r][c].NMult[i].y;
        u[i] = fptr->Data_t0.Velocity.x –
               ifptr->Velocity.x;
        v[i] = fptr->Data_t0.Velocity.y –
               ifptr->Velocity.y;
        p[i] = fptr->Data_t0.Pressure –
               ifptr->Pressure;
        d[i] = fptr->Data_t0.Density –
               ifptr->Density;
        s = sqrt(x[i]*x[i] + y[i]*y[i]) / HO_EFF;
        w[i] = 1-s;

        if(fptr->ParticleType == tptBOUNDARY)
            w[i] = 1;

        ifptr->AveV.x +=
               fptr->Data_t0.Velocity.x;
        ifptr->AveV.y +=
               fptr->Data_t0.Velocity.y;
        ifptr->AveP += fptr->Data_t0.Pressure;
        ifptr->AveR += fptr->Data_t0.Density;
}

ifptr->AveV.x /= NUM_NEIGHBORS;
ifptr->AveV.y /= NUM_NEIGHBORS;
ifptr->AveP /= NUM_NEIGHBORS;
ifptr->AveR /= NUM_NEIGHBORS;

if(doVelocity)
{
   fx = fy = fxx = fyy = 0;
   ComputeGradientsBiQuadraticNormalEquations(x,
                 y, u, w, &fx, &fy, &fxx, &fyy);

   Particle[r][c].Gradients.U.dx = fx;
   Particle[r][c].Gradients.U.dy = fy;
   Particle[r][c].Gradients.U.dxdx = fxx;
   Particle[r][c].Gradients.U.dydy = fyy;

   fx = fy = fxx = fyy = 0;
   ComputeGradientsBiQuadraticNormalEquations(x,
                 y, v, w, &fx, &fy, &fxx, &fyy);
   Particle[r][c].Gradients.V.dx = fx;
   Particle[r][c].Gradients.V.dy = fy;
   Particle[r][c].Gradients.V.dxdx = fxx;
   Particle[r][c].Gradients.V.dydy = fyy;
}

if(doPressure)
{
   fx = fy = fxx = fyy = 0;
   ComputeGradientsPlaneNormalEquations(x,
                    y, p, w, &fx, &fy);

   Particle[r][c].Gradients.P.dx = fx;
   Particle[r][c].Gradients.P.dy = fy;
}

if(doDensity)
{
   fx = fy = fxx = fyy = 0;
   ComputeGradientsPlaneNormalEquations(x,
                     y, d, w, &fx, &fy);

   Particle[r][c].Gradients.R.dx = fx;
   Particle[r][c].Gradients.R.dy = fy;
}

if((c <= 2) || (c >= NUM_COLS - 3))
// right (outflow) and left
// (inflow) BCs zero gradient...
{
        Particle[r][c].Gradients.U.dx = 0.0;
        Particle[r][c].Gradients.U.dy = 0.0;
        Particle[r][c].Gradients.U.dxdx = 0.0;
        Particle[r][c].Gradients.U.dydy = 0.0;
        Particle[r][c].Gradients.V.dx = 0.0;
        Particle[r][c].Gradients.V.dy = 0.0;
        Particle[r][c].Gradients.V.dxdx = 0.0;
        Particle[r][c].Gradients.V.dydy = 0.0;
        Particle[r][c].Gradients.P.dx = 0.0;
        Particle[r][c].Gradients.P.dy = 0.0;
        Particle[r][c].Gradients.R.dx = 0.0;
        Particle[r][c].Gradients.R.dy = 0.0;
}

if( (r <= 2) || (r >= NUM_ROWS - 3) )
// top and bottom BCs to prevent
// reflected pressure wave...
{
        Particle[r][c].Gradients.U.dx = 0.0;
        Particle[r][c].Gradients.U.dy = 0.0;
        Particle[r][c].Gradients.U.dxdx = 0.0;
        Particle[r][c].Gradients.U.dydy = 0.0;
        Particle[r][c].Gradients.V.dx = 0.0;
```

```
                                        Particle[r][c].Gradients.V.dy = 0.0;
                                        Particle[r][c].Gradients.V.dxdx = 0.0;
                                        Particle[r][c].Gradients.V.dydy = 0.0;

                                        Particle[r][c].Gradients.P.dx = 0.0;
                                        Particle[r][c].Gradients.P.dy = 0.0;
                                        Particle[r][c].Gradients.R.dx = 0.0;
                                        Particle[r][c].Gradients.R.dy = 0.0;
                                }
                        }
                }
        }
```

ComputeGradientsBiQuadraticNormalEquations

This function computes field value gradients using the DPM method for fitting bi-quadratic shape functions.

```
void    TWedgeModel::ComputeGradientsBiQuadraticNormalEquations(double*
        x, double* y, double* f, double* w, double* fx, double* fy,
        double* fxx, double* fyy)
{
        double          cA, cB, cD, cE, cF, cG, cH;
        double          cI, cJ, cK, cL, cM, cT, cU;
        double          cC, cO, cP, cQ, cR, cS, cN;
        double          fee[6][7], beta[6][7], gamma[6][7], theta[6][7];
        double          a[6];
        int             i;

        cA = cB = cD = cE = cF = cG = cH = 0;
        cI = cJ = cK = cL = cM = cT = cU = 0;
        cC = cO = cP = cQ = cR = cS = 0;

        for(i=0; i<NUM_NEIGHBORS; i++)
        {
                // summations involving only x and y...
                cA += w[i] * x[i];
                cB += w[i] * y[i];
                cD += w[i] * (x[i]*y[i]);
                cE += w[i] * (x[i]*x[i]*y[i]);
                cF += w[i] * (x[i]*y[i]*y[i]);
                cG += w[i] * (x[i]*x[i]*y[i]*y[i]);
                cH += w[i] * (x[i]*x[i]*x[i]);
                cI += w[i] * (y[i]*y[i]);
                cJ += w[i] * (x[i]*y[i]*y[i]*y[i]);
                cK += w[i] * (x[i]*x[i]*x[i]*y[i]);
                cL += w[i] * (x[i]*x[i]*x[i]*x[i]);
                cM += w[i] * (y[i]*y[i]*y[i]*y[i]);
                cN += w[i];
                cT += w[i] * (x[i]*x[i]);
        cU += w[i] * (y[i]*y[i]*y[i]);

        // additional summations involving f as well...
        cC += w[i] * f[i];
        cO += w[i] * (x[i]*f[i]);
        cP += w[i] * (y[i]*f[i]);
        cQ += w[i] * (x[i]*y[i]*f[i]);
        cR += w[i] * (x[i]*x[i]*f[i]);
        cS += w[i] * (y[i]*y[i]*f[i]);
}

        // find Ux, Uxx, Uy, Uyy
        fee[0][0] = cN;
        fee[0][1] = cA;
        fee[0][2] = cB;
        fee[0][3] = cT;
        fee[0][4] = cD;
        fee[0][5] = cI;
        fee[0][6] = cC;

        fee[1][0] = 0;
        fee[1][1] = cT*cN - cA*cA;
        fee[1][2] = cD*cN-cA*cB;
        fee[1][3] = cH*cN-cA*cT;
        fee[1][4] = cE*cN-cA*cD;
        fee[1][5] = cF*cN-cA*cI;
        fee[1][6] = cO*cN-cA*cC;

        fee[2][0] = 0;
        fee[2][1] = 0;
        fee[2][2] = (cI*cN-cB*cB)*(cT*cN-cA*cA)-
                    (cD*cN-cB*cA)*(cD*cN-cA*cB);
        fee[2][3] = (cE*cN-cB*cT)*(cT*cN-cA*cA)-
                    (cD*cN-cB*cA)*(cH*cN-cA*cT);
        fee[2][4] = (cF*cN-cB*cD)*(cT*cN-cA*cA)-
                    (cD*cN-cB*cA)*(cE*cN-cA*cD);
        fee[2][5] = (cU*cN-cB*cI)*(cT*cN-cA*cA)-
                    (cD*cN-cB*cA)*(cF*cN-cA*cI);
        fee[2][6] = (cP*cN-cB*cC)*(cT*cN-cA*cA)-
                    (cD*cN-cB*cA)*(cO*cN-cA*cC);

        fee[3][0] = 0;
        fee[3][1] = 0;
        fee[3][2] = (cT*cN-cA*cA)*(cE*cN-cT*cB)-
                    (cH*cN-cT*cA)*(cD*cN-cA*cB);
        fee[3][3] = (cT*cN-cA*cA)*(cL*cN-cT*cT)-
                    (cH*cN-cT*cA)*(cH*cN-cA*cT);
        fee[3][4] = (cT*cN-cA*cA)*(cK*cN-cT*cD)-
                    (cH*cN-cT*cA)*(cE*cN-cA*cD);
        fee[3][5] = (cT*cN-cA*cA)*(cG*cN-cT*cI)-
                    (cH*cN-cT*cA)*(cF*cN-cA*cI);
        fee[3][6] = (cT*cN-cA*cA)*(cR*cN-cT*cC)-
                    (cH*cN-cT*cA)*(cO*cN-cA*cC);
```

```
fee[4][0] = 0;
fee[4][1] = 0;
fee[4][2] = (cT*cN-cA*cA)*(cF*cN-cD*cB)-
            (cE*cN-cD*cA)*(cD*cN-cA*cB);
fee[4][3] = (cT*cN-cA*cA)*(cK*cN-cD*cT)-
            (cE*cN-cD*cA)*(cH*cN-cA*cT);
fee[4][4] = (cT*cN-cA*cA)*(cG*cN-cD*cD)-
            (cE*cN-cD*cA)*(cE*cN-cA*cD);
fee[4][5] = (cT*cN-cA*cA)*(cJ*cN-cD*cI)-
            (cE*cN-cD*cA)*(cF*cN-cA*cI);
fee[4][6] = (cT*cN-cA*cA)*(cQ*cN-cD*cC)-
            (cE*cN-cD*cA)*(cO*cN-cA*cC);

fee[5][0] = 0;
fee[5][1] = 0;
fee[5][2] = (cT*cN-cA*cA)*(cU*cN-cI*cB)-
            (cF*cN-cI*cA)*(cD*cN-cA*cB);
fee[5][3] = (cT*cN-cA*cA)*(cG*cN-cI*cT)-
            (cF*cN-cI*cA)*(cH*cN-cA*cT);
fee[5][4] = (cT*cN-cA*cA)*(cJ*cN-cI*cD)-
            (cF*cN-cI*cA)*(cE*cN-cA*cD);
fee[5][5] = (cT*cN-cA*cA)*(cM*cN-cI*cI)-
            (cF*cN-cI*cA)*(cF*cN-cA*cI);
fee[5][6] = (cT*cN-cA*cA)*(cS*cN-cI*cC)-
            (cF*cN-cI*cA)*(cO*cN-cA*cC);

gamma[3][3] = fee[2][2]*fee[3][3] - fee[3][2]*fee[2][3];
gamma[3][4] = fee[2][2]*fee[3][4] - fee[3][2]*fee[2][4];
gamma[3][5] = fee[2][2]*fee[3][5] - fee[3][2]*fee[2][5];
gamma[3][6] = fee[2][2]*fee[3][6] - fee[3][2]*fee[2][6];

gamma[4][3] = fee[2][2]*fee[4][3] - fee[4][2]*fee[2][3];
gamma[4][4] = fee[2][2]*fee[4][4] - fee[4][2]*fee[2][4];
gamma[4][5] = fee[2][2]*fee[4][5] - fee[4][2]*fee[2][5];
gamma[4][6] = fee[2][2]*fee[4][6] - fee[4][2]*fee[2][6];

gamma[5][3] = fee[2][2]*fee[5][3] - fee[5][2]*fee[2][3];
gamma[5][4] = fee[2][2]*fee[5][4] - fee[5][2]*fee[2][4];
gamma[5][5] = fee[2][2]*fee[5][5] - fee[5][2]*fee[2][5];
gamma[5][6] = fee[2][2]*fee[5][6] - fee[5][2]*fee[2][6];

theta[4][4] = gamma[3][3]*gamma[4][4] –
              gamma[4][3]*gamma[3][4];
theta[4][5] = gamma[3][3]*gamma[4][5] -
              gamma[4][3]*gamma[3][5];
theta[4][6] = gamma[3][3]*gamma[4][6] –
              gamma[4][3]*gamma[3][6];

theta[5][4] = gamma[3][3]*gamma[5][4] –
              gamma[5][3]*gamma[3][4];
theta[5][5] = gamma[3][3]*gamma[5][5] –
              gamma[5][3]*gamma[3][5];
theta[5][6] = gamma[3][3]*gamma[5][6] –
              gamma[5][3]*gamma[3][6];
```

```
                beta[5][5] = theta[4][4]*theta[5][5] –
                             theta[5][4]*theta[4][5];
                beta[5][6] = theta[4][4]*theta[5][6] –
                             theta[5][4]*theta[4][6];

                a[5] = beta[5][6] / beta[5][5];
                a[4] = (theta[4][6] - a[5]*theta[4][5])/theta[4][4];
                a[3] = (gamma[3][6] - a[4]*gamma[3][4] –
                       a[5]*gamma[3][5])/gamma[3][3];
                a[2] = (fee[2][6] - a[3]*fee[2][3] - a[4]*fee[2][4] –
                       a[5]*fee[2][5])/fee[2][2];
                a[1] = (fee[1][6] - a[2]*fee[1][2] - a[3]*fee[1][3] –
                       a[4]*fee[1][4] - a[5]*fee[1][5])/fee[1][1];
                a[0] = (fee[0][6] - a[1] * fee[0][1] - a[2] * fee[0][2] –
                       a[3] * fee[0][3] - a[4] * fee[0][4] –
                       a[5] * fee[0][5]) / fee[0][0];

                *fx = a[1];
                *fxx = 2 * a[3];
                *fy = a[2];
                *fyy = 2 * a[5];
}
```

ComputeGradientsPlaneNormalEquations

This function computes field value gradients using the DPM method for fitting plane equation shape functions.

```
void    TWedgeModel::ComputeGradientsPlaneNormalEquations(double* x,
            double* y, double* f, double* w, double* fx, double* fy)
{

        double  A, B, C, D, F, G, H, K, L;
        double  M, N, P;
        int     i;
        double  a, b, c;

        a = b = c = 0;
        M = N = P = 0;
        A = B = C = D = F = G = H = K = L = 0;

        for(i=0; i<NUM_NEIGHBORS; i++)
        {
                A += w[i];
                B += w[i] * x[i];
                C += w[i] * y[i];
                D += w[i] * f[i];
                F += w[i] * x[i] * x[i];
                G += w[i] * y[i] * x[i];
                H += w[i] * x[i] * f[i];
        K += w[i] * y[i] * y[i];
        L += w[i] * y[i] * f[i];
}

M = (F - ((B*B)/A));
N = (H - ((B * D) / A));
P = ((B*G)/(A*M)) - ((B*B*C)/(A*A*M)) - (C/A);

c = (L - (C*D)/A + (C*B*N)/(A*M) - (G*N)/M) / ((P*C) - (G*G)/M +
    (B*C*G)/(A*M) + K);
b = (N/M) - (G * c)/M + (B*C*c)/(A*M);
a = (D - B*b - C*c)/A;

*fx = b;
*fy = c;
                            }
```

Vita

David Bourg is a native Louisianan born in 1968 to Robert and Barbara Bourg. While born in New Orleans, David grew up in the then rural town of Destrehan where he enjoyed all the typical activities of southern youth including fishing, hunting, and carefree exploration of local swamplands. Just months before enlisting in the US Navy at the age of 18, David spontaneously enrolled in Naval Architecture and Marine Engineering at the University of New Orleans. He did not at that time know what Naval Architecture was and selected it simply because the curriculum included the work “architecture.” David was a fan of Frank Lloyd Wright style civil architecture and assumed Naval Architecture was close enough. Little did he know that Naval Architecture was a very specialized and fascinating field within which he would find vast fields of intellectual produce awaiting harvest.

As a youth, David spent countless hours fashioning rafts and boats at both model and full scale and he found the field of Naval Architecture a natural calling. He earned his Bachelor of Science degree in Naval Architecture and Marine Engineering from the University of New Orleans in 1992 shortly followed by his Master of Sciences degree in 1993. David took a break from his studies to build a career in the field working for various local shipyards and design firms all the while teaching as an Adjunct Professor at the School of Naval Architecture and Marine Engineering.

After branching out into the realm of video game development, physics-based simulations, and artificial intelligence David authored two books on these subjects and then returned to his studies in Naval Architecture at the University of New Orleans. While working on his PhD David authored his third book on scientific and engineering computing. He is currently a Managing Partner for MiNO Marine LLC, a Naval Architecture and marine professional services company which he founded.