A Meshfree Method for the Poisson Equation with 3D Wall-Bounded Flow Application

by

Anna Vasilyeva

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Aeronautics and Astronautics
August 19, 2010

Certified by .....
Ahmed F. Ghoniem
Ronal C. Crane (1972) Professor
Thesis Supervisor

Accepted by .....
Eytan H. Modiano
Chair, Graduate Program Committee

A Meshfree Method for the Poisson Equation with 3D Wall-Bounded Flow Application

by

Anna Vasilyeva

Submitted to the Department of Aeronautics and Astronautics
on August 19, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

The numerical approximation of the Poisson equation can often be found as a sub-problem to many more complex computations. In the case of Lagrangian approaches of flow equations, the Poisson equation often needs to be solved on an irregular point distribution. Currently, mainly unstructured mesh-based approaches are used. Meshfree methods present a way to approximate differential operators on unstructured point clouds without the need for mesh generation. In this thesis, a 3d meshfree finite difference Poisson solver is presented. Its performance has been studies based on numerical convergence, parallel efficiency, and computational cost. Practical application of the solver is presented in a simulation of a potential flow field in a wall-bounded domain.

Thesis Supervisor: Ahmed F. Ghoniem
Title: Ronal C. Crane (1972) Professor

Acknowledgments

I would like to express my gratitude to Professor Ghoniem for his support of my research during the last two years. I am very grateful for being given the opportunity to work at the Reacting Gas Dynamics Laboratory and for the help and advice I have received from him.

I would also like to thank my academic adviser Professor Radovitzky of the Aero-Asto department, Barbara Lechner, Marie Stuppard, and Beth Marois of the department's administrative staff for their kind support and useful advice.

I am grateful to my colleagues at the Reacting Gas Dynamics Lab, especially to Fabrice Schlegel for guiding me along my research and offering helpful recommendations for my work. A special thank you to Dr. Jean-Christophe Nave for his help on various aspects of my project. A thank you also to Lorraine Rabb for her administrative assistance.

Finally, I want to acknowledge my parents, brothers, and all my friends who have offered me their support and encouragement throughout my time at MIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the need to solve larger and more complex computational problems grows, so does the need to efficiently discretize domains used in these problems. One way to approach this problem is with meshfree methods. Meshfree methods are useful in that they eliminate the need for pre-specified connectivity of points and thus alleviate some of the complications that come with discretizing a domain using a computational mesh. They can also naturally handle issues with complex boundaries, large deformations, and domain discontinuities.

Vortex methods [7] are a meshless approach to solving the Navier-Stokes equations, and are useful in simulating flows at high Reynolds numbers with areas of strong vorticity. These methods are based on a Lagrangian formulation, meaning that the vortex elements advect with the local velocity. They have the advantage of not needing to conform to the CFL condition and having minimal numerical dissipation. However, a mesh can usually still reappear in stages like interpolation, or determining velocity on a mesh and then interpolating it to domain particles such as in the vortex in cell method [8]. This offsets some of the benefits of the Lagrangian vortex method as it introduces numerical diffusion.

There is a need to develop and utilize efficient meshfree approaches in CFD. Meshfree Poisson solvers have been implemented to simulate incompressible flows by the Lagrangian particle method, particularly in the projection step [15]. This is done when a standard finite difference Poisson solver cannot be directly applied.

Furthermore, moving least squares method for solving fluid dynamic problems have been studied [4, 10], where spatial derivatives are approximated using an arbitrary surrounding cloud of points.

The main focus of this thesis is to develop an efficient meshfree finite difference Poisson solver that can be used in conjunction with vortex methods and preserve the mesh-free nature of these methods. The solver is based on least squares approximation and is applicable to problems where generating a mesh is complicated or costly, specifically complex time-dependent geometries. Instead of a mesh, it approximates differential operators by establishing meshfree neighborhood relations. The solver thus allows us to treat velocity in a vorticity simulation in a completely meshfree way.

We begin by formulating the meshfree finite difference in chapter 2 based on Taylor expansion. Chapter 3 explores in depth a meshfree finite difference solver in 2d and 3d through various tests of numerical convergence, parallel efficiency, iterative solvers, neighbor selection approaches, and computational cost. An implementation of the solver in a confined flow simulation is described in chapter 4. This is followed by some concluding remarks in chapter 5.

Chapter 2

Numerical Method for the Poisson Equation

In this chapter we formulate a meshfree finite difference numerical scheme for solving the Poisson equation using a least squares approximation. We consider the following Poisson equation inside a domain Ω

$$\Delta u = f \quad \text{in } \Omega \tag{2.1}$$

with Dirichlet boundary conditions

$$u = g \quad \text{on } \Gamma, \tag{2.2}$$

or Neumann boundary conditions

$$\frac{\partial u}{\partial n} = h \quad \text{on } \Gamma. \tag{2.3}$$

Solving this system with finite difference converts the problem into a linear system of equations

$$A \cdot \mathbf{u} = \mathbf{f}, \tag{2.4}$$

where A is a matrix containing the meshfree finite difference stencils and $\mathbf{u}$ is the vector of the approximate solutions.

2.1 Meshfree Finite Difference Stencil Formulation

Considering a function $u \in C^l$, we want to approximate differential operators by finitely many function values

$$\frac{\partial^k u}{\partial x^k}(x_0) \approx \sum_{i=0}^{m} \alpha_i u(x_i) \qquad (k \leq l), \tag{2.5}$$

where the vector of coefficients $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_m)$ is the finite difference stencil.

By using Taylor expansion, the function value of each point x_0 in a meshfree domain can be approximated using the function values of its neighboring points x_i. In 1d, defining the distance vector $\bar{x}_i = x_i - x_0$, the Taylor expansion around the point x_i is

$$u(x_i) = u(x_0) + u_x(x_0) + u_x(x_0) \cdot \bar{x}_i + \frac{1}{2} u_{xx}(x_0) \cdot \bar{x}_i^2 + e_i, \tag{2.6}$$

where e_i is the error in the expansion at the point x_i. A linear combination of the above expansion with stencil coefficients $(\alpha_0, \alpha_1, \ldots, \alpha_m)$ gives

$$\sum_{i=0}^{m} \alpha_i u(x_i) = u(x_0) \sum_{i=0}^{m} \alpha_i + u_x(x_0)(\sum_{i=0}^{m} \alpha_i \bar{x}_i) + u_{xx}(x_0)(\frac{1}{2} \sum_{i=0}^{m} \alpha_i \bar{x}_i^2) + e_i. \tag{2.7}$$

Matching coefficients, equation (2.7) approximates the second derivative if

$$\sum_{i=0}^{m} \alpha_i = 0, \quad \sum_{i=0}^{m} \alpha_i \bar{x}_i = 0, \quad \sum_{i=0}^{m} \alpha_i \bar{x}_i^2 = 2. \tag{2.8}$$

The above constraints on the stencil form a linear system

$$V \cdot \alpha = \mathbf{b}, \tag{2.9}$$

where a 1d Vandermonde matrix V and vector $\mathbf{b}$ take the following form

$$V = \begin{pmatrix} 1 & \dots & 1 \\ \bar{x}_0 & \dots & \bar{x}_m \\ \bar{x}_0^2 & \dots & \bar{x}_m^2 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}. \tag{2.10}$$

In general, for m neighboring points and k constraints defined by (2.8), there is one unique stencil solution if $m = k$, an over-determined system with multiple stencil solutions if $m > k$, and no stencil solution if $m < k$. For a consistent approximation of the second derivative in 3d, for example, at least 10 neighbors are needed to determine the stencil.

2.1.1 Least Squares Method

The finite difference stencil is computed using a weighted quadratic minimization formulation

$$J = \sum_{i=0}^{n} \frac{\alpha_i^2}{w_i} \tag{2.11}$$

subject to constraints in (2.9), which we rewrite as

$$g_i(\boldsymbol{\alpha}) = \sum_{j=0}^{n} v_{ij}\alpha_j - b_i = 0. \tag{2.12}$$

The weights w_i are assigned to each neighboring point in a way such that the neighboring points closer to the central point in the meshfree domain have higher weight values than those further away. The minimization of J can be obtained using Lagrange multipliers

$$\nabla J(\boldsymbol{\alpha}) = \sum_{i=0}^{n} \lambda_i \nabla g_i(\boldsymbol{\alpha}), \tag{2.13}$$

or

$$2\sum_{i=0}^{m} \frac{\alpha_i}{w_i} = \sum_{j=0}^{n} \lambda_j v_{ij}. \tag{2.14}$$

In matrix form

$$2W^{-1} \cdot \boldsymbol{\alpha} = V^T \cdot \boldsymbol{\lambda}, \tag{2.15}$$

where W is a diagonal weight matrix containing the given weights of each neighbor point for a specific central point

$$W = \begin{pmatrix} w_0 & 0 & \dots & 0 \\ 0 & w_1 & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}. \tag{2.16}$$

Solving (2.15) for $\boldsymbol{\alpha}$

$$\boldsymbol{\alpha} = \frac{1}{2} W V^T \cdot \boldsymbol{\lambda}. \tag{2.17}$$

Putting (2.17) into (2.9) gives

$$V \cdot (\frac{1}{2} W V^T \cdot \boldsymbol{\lambda}) = \mathbf{b} \tag{2.18}$$

and solving for $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda} = 2(VWV^T)^{-1} \cdot \mathbf{b}. \tag{2.19}$$

Putting (2.19) into (2.17) yields the stencil vector

$$\boldsymbol{\alpha} = (V^T W V)^{-1}(V^T W) \cdot \mathbf{b}. \tag{2.20}$$

The stencil vector corresponding to the point x_i makes up the i-th row of the system matrix A in (2.4). The system can then be solved for the approximate values of $u(x_i)$.

2.1.2 Higher Order Approximations and Neumann Boundary Points

Both Neumann boundary points defined in (2.3) and higher order approximations can be obtained by making modifications to the Taylor expansion and constraint definitions in (2.8). If the constraints are set up to approximate the first derivative as in the case of Neumann boundary points, the Taylor expansion needs to only include the first two terms of (2.6), with the second order term included in the error of the

expansion. In this case, the constraints are given by

$$\sum_{i=0}^{m} \alpha_i = 0, \quad \sum_{i=0}^{m} \alpha_i \bar{x}_i = \mathbf{n}, \tag{2.21}$$

where $\mathbf{n}$ is the normal vector for the directional derivative $\frac{\partial}{\partial \mathbf{n}}$. In 1d, the constraint system would read

$$V = \begin{pmatrix} 1 & \dots & 1 \\ \bar{x}_0 & \dots & \bar{x}_m \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.22}$$

The above formulation along with the one presented in (2.10) guarantees at least first order accurate approximations for the Poisson equation with either Dirichlet or Neumann boundary conditions. Higher order approximations can be obtained by including higher order terms in the Taylor expansion and setting them to zero in the stencil constraints. For example, if the constraints in (2.10) were expanded to include fourth order terms, the Vandermonde matrix V and vector $\mathbf{b}$ would be as follows

$$V = \begin{pmatrix} 1 & \dots & 1 \\ \bar{x}_0 & \dots & \bar{x}_m \\ \bar{x}_0^2 & \dots & \bar{x}_m^2 \\ \bar{x}_0^3 & \dots & \bar{x}_m^3 \\ \bar{x}_0^4 & \dots & \bar{x}_m^4 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}. \tag{2.23}$$

However, it can be seen from the above constraints that $\sum_{i=0}^{m} \bar{x}_i^4 \alpha_i = 0$ is only satisfied for positive stencil entries if $\alpha_i = 0$, which leads to an inconsistent approximation. Imposing higher order conditions can thus lead to both positive and negative stencil entries, which is undesirable since this worsens the stability of the system. This also means that second order approximation of Neumann boundary points will never lead to positive stencils due to the constraints $\sum_{i=0}^{m} \bar{x}_i^2 \alpha_i = 0$. Further discussion on positive stencils can be found in Seibold [22].

Additionally, it should be noted that the constraint system can be reduced if the first constraint in (2.8) is computed by the relation $\alpha_0 = -\sum_{i=1}^{m} \alpha_i$. The reduced

system for approximating the Poisson equation in 1d is given by

$$V = \begin{pmatrix} \bar{x}_1 & \dots & \bar{x}_m \\ \bar{x}_1^2 & \dots & \bar{x}_m^2 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}. \tag{2.24}$$

With the reduced system in 3d, for instance, the invertible V matrix is 9×9 instead of 10×10. This does not, however, reduce the computational costs by much. Reducing the V matrix also keeps the convergence rate the same.

2.2 Weight Function

The selection of the weight function can be quite arbitraty. Generally, it is desired to select a function that will allow neighboring points far away from the central point to carry little or no weight at all in the stencil calculation; in this sense, the weight function restricts the number of points used in the approximation of differential operators. In this thesis, we look at an exponentially decaying function

$$w(x_i - x_0) = exp(-\gamma||x_i - x_0||^2), \tag{2.25}$$

where γ is an adjustable constant. The approximation errors in our meshfree finite difference formulation depend on γ. For instance, for the Poisson equation in 1d with a regular distribution of points, as the mesh size h decreases, the parameter γ should increase so that the computed system matrix A will always consist of the standard three point finite difference stencil $\frac{1}{h^2}(1, -2, 1)$. The same adjustment to the γ parameter applies for approximations in higher dimensions. If γ is not adjusted, the weight function might not decay fast enough and too many points will be included as neighbors with high weight values. This will result in both positive and negative stencil entries in the system matrix, worsening stability and causing oscillations in the approximate solution.

2.2.1 Poisson Weight Function Test Problem

To examine the weight function, we consider a one dimensional Poisson problem

$$\begin{aligned} -u_{xx} &= f \quad \text{in }]0,1[\\ u &= 0 \quad \text{on } [0,1], \end{aligned} \tag{2.26}$$

with $f(x) = \sin(\phi(x))(\phi_x(x))^2 - \cos(\phi(x))\phi_{xx}(x)$, where $\phi(x) = 9\pi x^2$. The problem in (2.26) has a unique solution

$$u(x) = \sin(9\pi x^2). \tag{2.27}$$

We use equation (2.25) as the weight function and calculate the weight for every point in the domain. Figure 2.1 demonstrates the stability problem of having too many neighboring points with high weight values. It is therefore appropriate to use a weight function that depends on the average mesh size h as well as distance between the neighboring points and central point. We adjust the weight function to the following

$$w(x_i - x_0) = exp(-\mu(\frac{||x_i - x_0||}{h})^2). \tag{2.28}$$

Two important factors for an accurate approximation are the global error in the approximation and the condition number of the final system matrix. A low condition number indicates better accuracy of the linear equation solution from matrix inversion. If we take the approximation error to be the error in the maximum norm defined by

$$||\mathbf{e}||_\infty = max_{i=1}^n |e_i|, \tag{2.29}$$

where A is the system matrix in (2.4), then Figure 2.2 shows the trends in error and conditon number of the system matrix for a 380 point least squares system with varying parameter μ.

Based on Figure 2.2, the following can be noted:

- The approximation error decreases with an increasing μ while the condition number increases.
- The best μ values for the weight function seem to be between 1 and 2.
- Even though the condition number increases with increasing μ, the system matrices for larger μ values stay reasonably conditioned.
- Both the approximation errors and condition numbers level off with a large enough μ value.

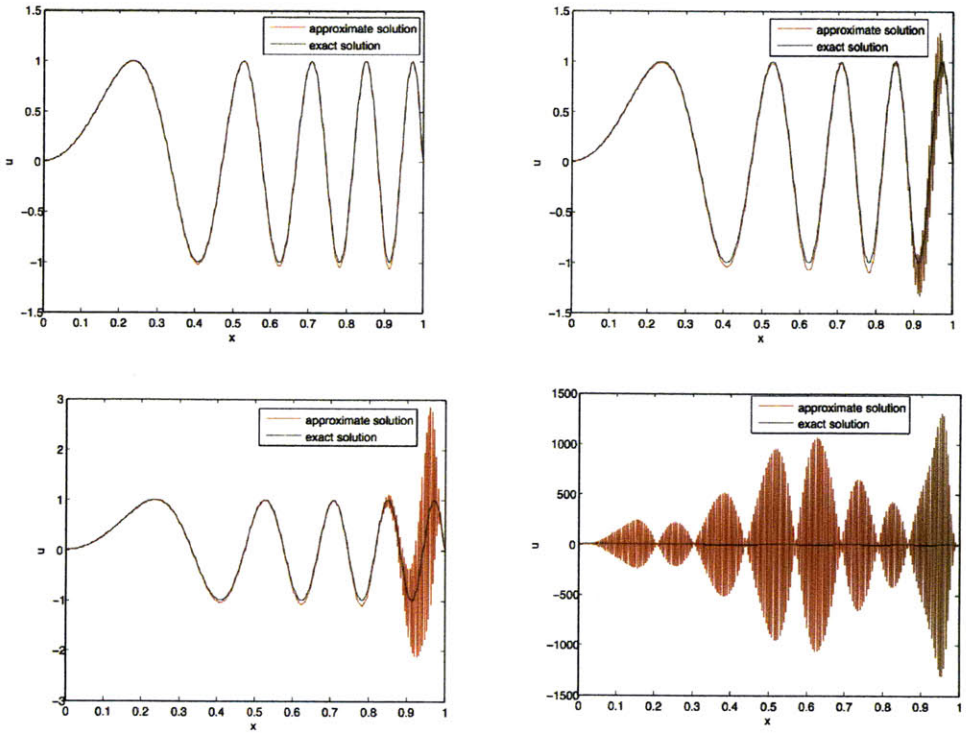


Figure 2-1: Evolution of a 380 point least squares system approximate solution with a decreasing γ and thus increasing weights

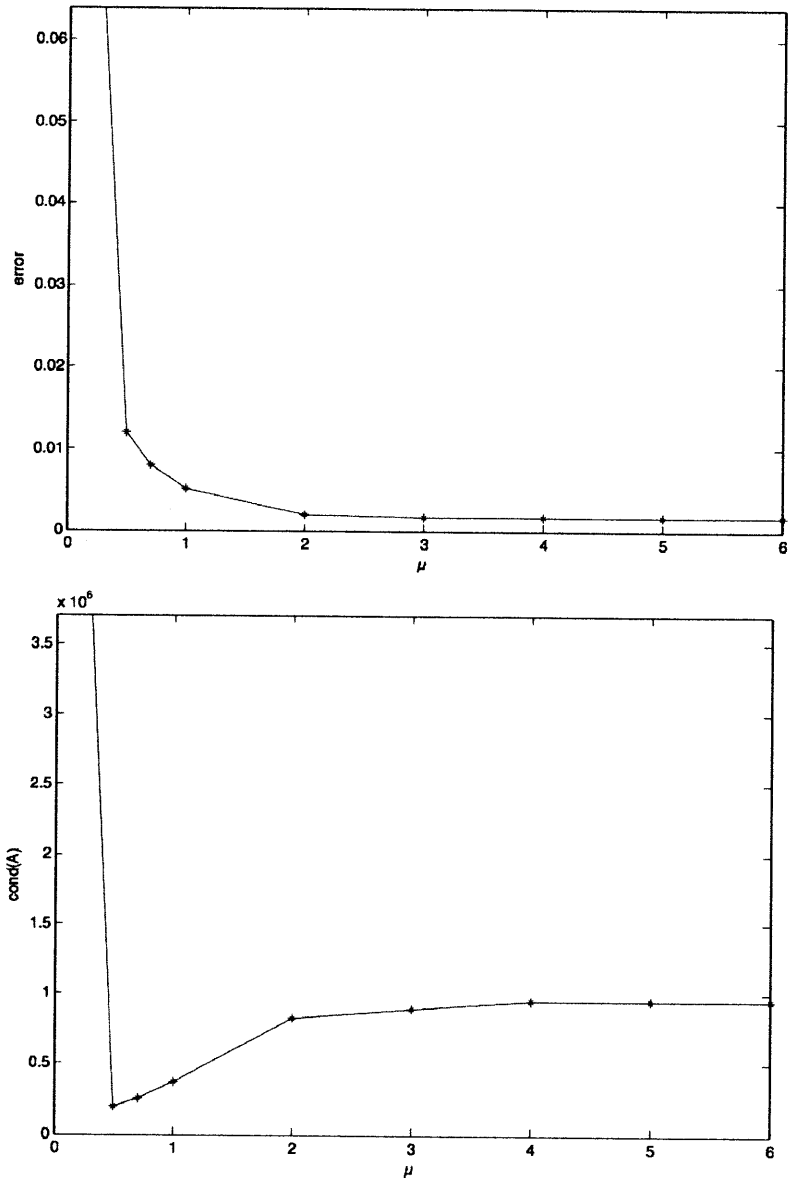


Figure 2-2: Error and system matrix condition number trends for 380 point least squares system

Chapter 3

Numerical Tests

In this chapter we investigate the numerical convergence, computational performance, and parallel efficiency of the meshfree finite difference approach described in chapter 2. We look at a Poisson problem formulation in 2d and 3d. In 2d we examine the numerical convergence of a Dirichlet and Neumann boundary problems and several neighbor selection approaches. For a 3d system, since the global linear system in finite difference can often pose a computational bottleneck, we examine various linear solvers and preconditioners provided by the fully parallel toolkit for scientific computation PETSc library [1, 2] and parallel performance. Furthermore, we consider the computational effort of the method and compare the meshfree approach to standard finite difference.

3.1 Numerical Convergence

We use the meshfree solver to approximate the Laplace operator in a circular and spherical domains. The Laplace equation in three-dimensional spherical coordinates is

$$\Delta u(r,\theta,\phi) = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial u}{\partial r}) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta\frac{\partial u}{\partial\theta}) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2 u}{\partial\phi^2} = 0 \qquad (3.1)$$

If we take into account spherical symmetry, then $\frac{\partial}{\partial\theta} = 0$ and $\frac{\partial}{\partial\phi} = 0$. We investigate the numerical convergence of the meshfree solver by considering the following Dirichlet

boundary value problem in 3d

$$\begin{aligned} \frac{1}{r^2}\frac{\partial}{\partial r}(r^2\frac{\partial u}{\partial r}) = 0 \quad \text{on } r \in (0.2, 1) \\ u(0.2) = 1, \quad u(1) = 0. \end{aligned} \tag{3.2}$$

The unique solution to this problem is

$$u(r) = \frac{1}{4r} - \frac{1}{4}. \tag{3.3}$$

Following the setup and boundary conditions in (3.2) but using polar coordinates, the Laplace equation in 2d is reduced to

$$\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial u}{\partial r}) = 0, \tag{3.4}$$

with a unique solution

$$u(r) = \frac{1}{\ln(0.2)}\ln(r). \tag{3.5}$$

Figure 3.1 shows the 2d and 3d computational domains described by this problem. Two cases are considered for each domain: regular distribution of both interior and boundary points and nonuniform distribution of interior points. The regular distribution is generated by placing equidistantly spaced points on concentric circles between the two boundaries. The nonuniform distribution is generated by taking the regular grid and introducing a small perturbation to every interior point. This guarantees that the points remain somewhat distributed throughout the domain without any major gaps but do not lie on any set grid. The points on the boundaries are kept evenly distributed in both cases.

Figure 3.2 shows a solution of a 2,828 point 2d system. We use the weight function defined in (2.28), with the parameter $\mu = 2$. The number of neighbors for each central point is kept fixed to the 6 closest neighbors, guaranteeing a consistent Laplace approximation in 2d.

Based on the Taylor expansion in (2.6), first order convergence is expected for the approximation of this Laplacian. However, systems with enough average symmetry

actually achieve second order convergence. Figure 3.3 shows second order convergence for the regular grid in both 2d and 3d and first order convergence for the nonuniform grid case. For the error of the nonuniform grid case, an average of 5 runs are taken along with the maximum and minimum error over the 5 runs. We use the error in the maximum norm as defined by (2.29). For the regular grid, the error is plotted against the average mesh size while for the nonuniform grid it is shown against the number of interior points. A reference line is plotted along with the error. The reference line is of slope 2 for second order convergence and of slope 1 for first order convergence. The 2d system ranges from 198 to 2,828 total points, while for the 3d system we use 6,272 to 856,570 total points to test error convergence.

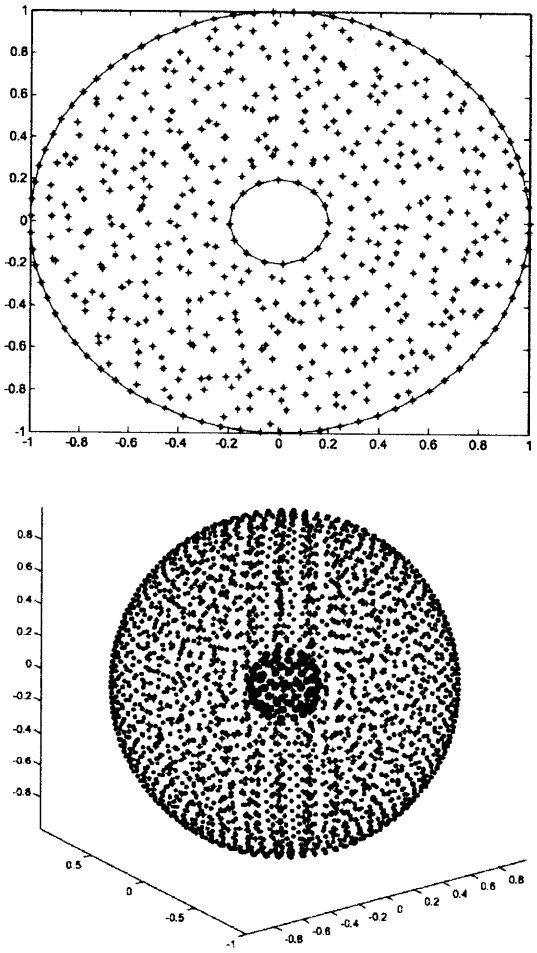


Figure 3-1: Computational domains for 2d and 3d systems with nonuniform distribution of interior points

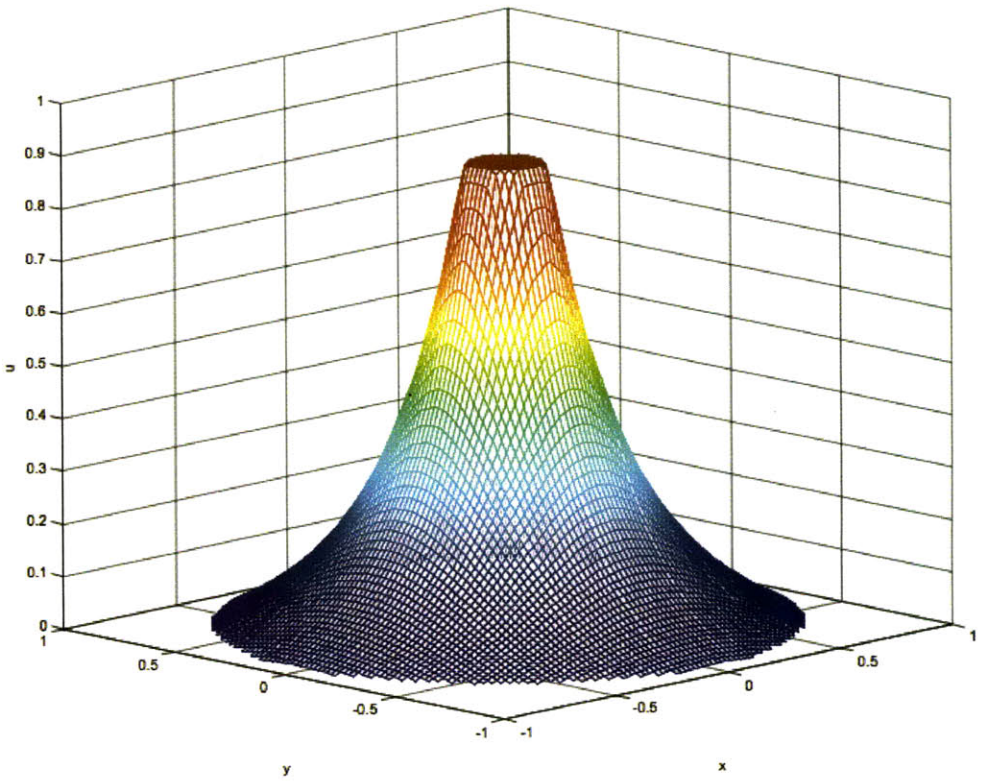


Figure 3-2: Solution of a 2d system

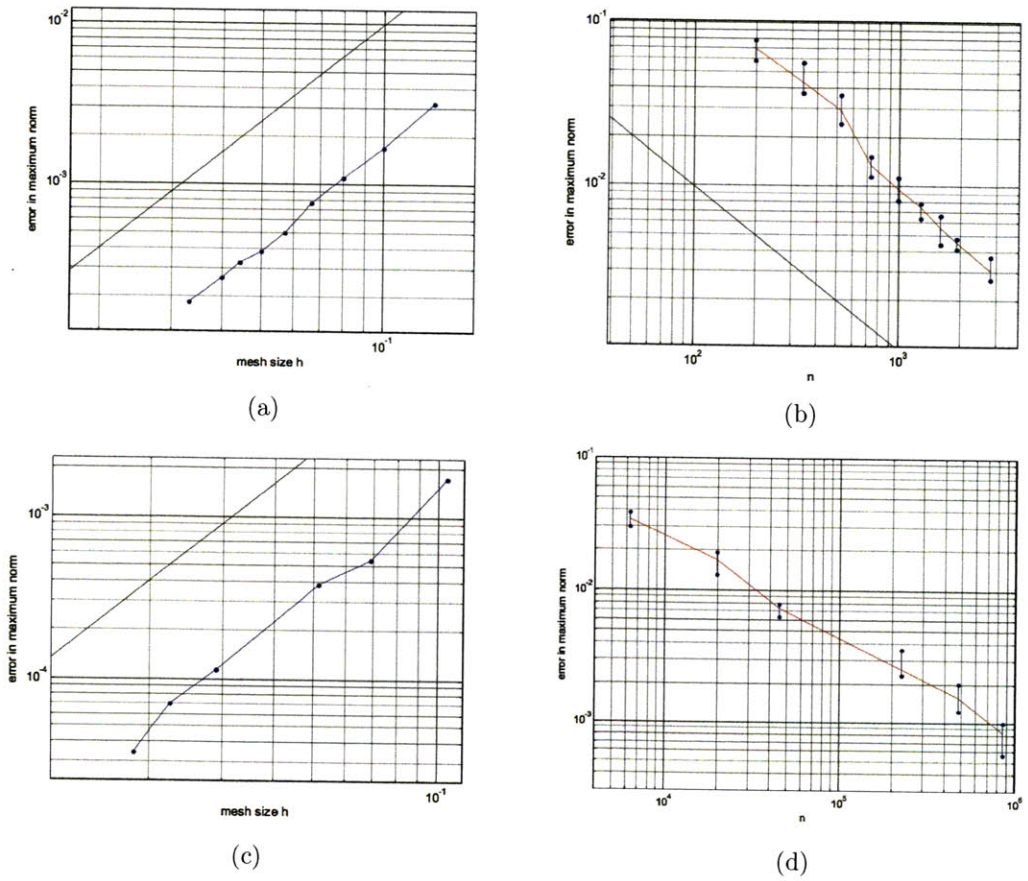


Figure 3-3: Error convergence of 2d and 3d systems - regular grid in 2d (a) and 3d (c), nonuniform grid in 2d (b) and 3d (d)

3.1.1 Neumann Boundary Points

To test the numerical convergence for Neumann boundary conditions, we discretize a unit square domain in 2d. We consider the following 2d Neumann boundary value problem

$$\begin{aligned} \Delta u &= -cos\pi x \quad \text{in } (0,1)\times(0,1) \\ \frac{\partial u}{\partial \mathbf{n}} &= 0 \quad \text{on } x=0, y=0, x=1, y=1. \end{aligned} \tag{3.6}$$

The unique solution to this problem is

$$u(x,y) = \frac{1}{\pi^2}\cos\pi x. \tag{3.7}$$

Figure 3.4 shows the solution of a 1,286 point system. The Neumann boundary conditions yield first order convergence when the system of constraints for the finite difference stencil is represented by (2.22). When higher order terms are included in the Taylor expansion, the solution yields higher order convergence. This can be seen in Figure 3.5 where first order convergence is achieved when higher terms are not included in the approximation and second order when one extra term is included in the Taylor expansion. Figure 3.5 also includes a reference line of slope 1 for first order convergence and slope 2 for second order convergence. The range of points is taken between 285 and 2,597. However, as mentioned earlier, although higher order approximations yield higher order convergence rates, they also result in an increase in the variation between positive and negative stencil values in the system matrix. Ideally, negative stencil values only appear on the diagonal of the system matrix, corresponding to the central points. Table 3.1 shows the distribution of negative stencil entries for systems of various sizes that either include or do not include higher order terms in the Taylor expansion. This does not include the central stencil entry. When dealing with large and complex problems, a high number of negative stencil entries can greatly affect the convergence of the system matrix to a solution.

number of points	no higher terms	higher terms
221	3	0
437	7	0
672	13	7
1152	22	10
2597	43	17
4485	57	23

Table 3.1: Distribution of the number of negative stencil entries in the system matrix

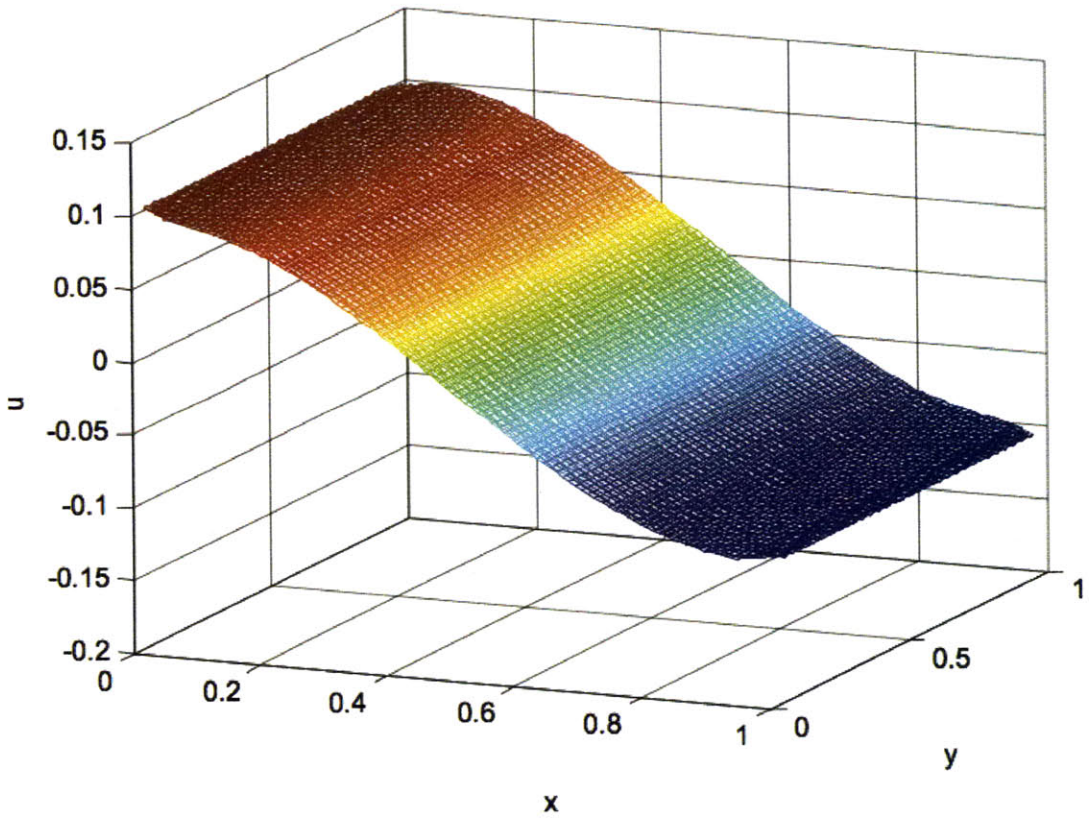
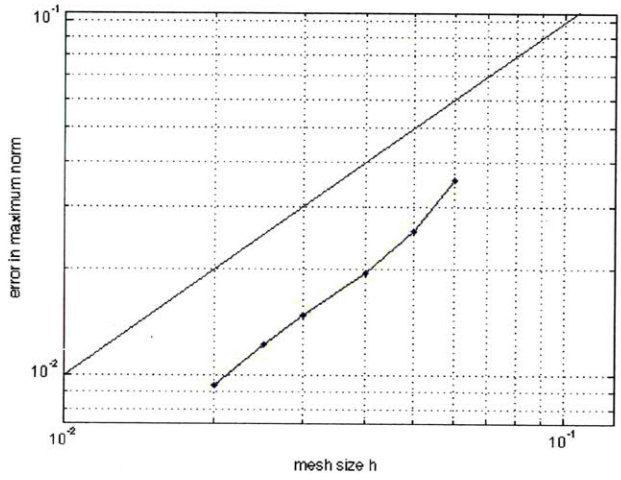
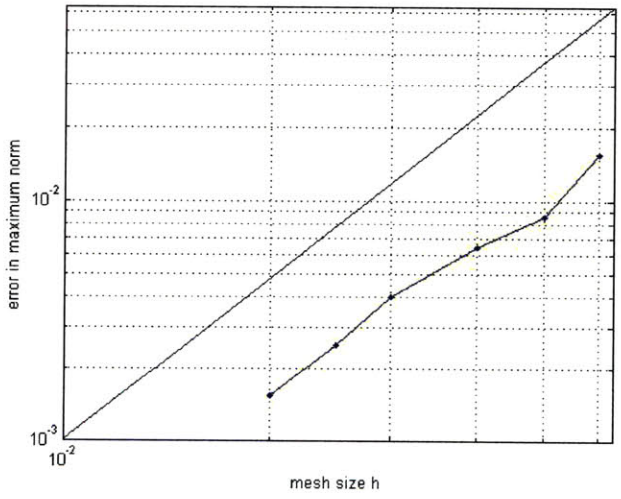


Figure 3-4: Solution of a 2d Neumann boundary value problem



(a)



(b)

Figure 3-5: Error convergence for first order approximation (a) and second order approximation (b)

3.1.2 Derivative Approximation

The goal with our meshfree solver is to be able to approximate the derivate of a function from its Poisson formulation. We therefore also analyze the numerical convergence of derivate approximation of u from (3.2). The derivative approximation is then based on

$$\frac{\partial u}{\partial \mathbf{n}} = A \cdot \mathbf{u}, \tag{3.8}$$

where A is the system matrix containing the stencil entries for a derivative approximation. We use a regular grid defined for (3.4) to test error convergence for this case in 2d. The error convergence using error in the maximum norm for a derivative approximation is shown in Figure 3.6, along with a reference line of slope 1. The system matrix with the stencils is based on second order approximation of the derivative. This derivative approximation yields a first order convergence, losing one order from the second order convergence observed for the approximation of the function value $\mathbf{u}$ in Figure 3.3(a).

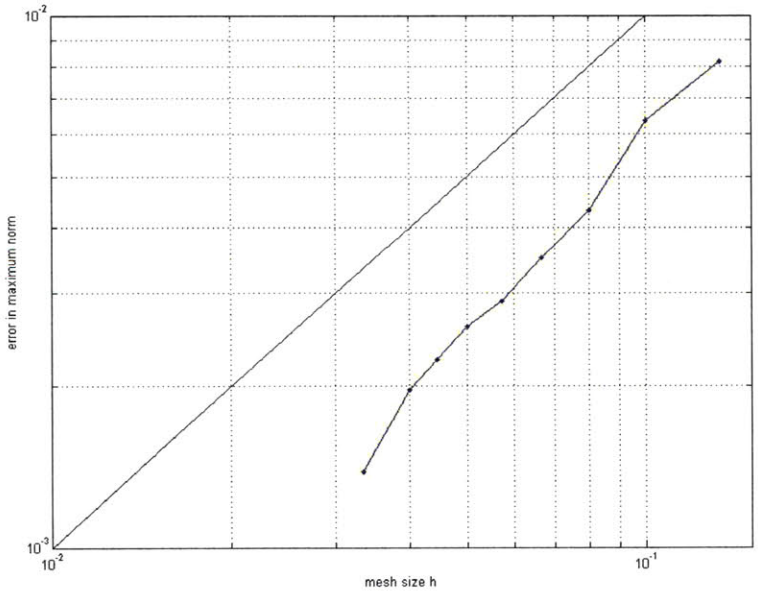


Figure 3-6: Error convergence of a 2d system for a derivative approximation

3.2 Neighbor Selection

Neighbor selection is an important step in setting up the stencil vector for each point in finite difference. There are various ways to define neighborhood criteria in mehsfree methods. Several such methods have been previously reviewed [16, 19, 21]. The key is to understand the geometry of the domain and the point distribution of the problem in order to select the best approach. Sometimes it is necessary to utilize more than one option within the same problem. We look at two ways of selecting neighbors and test out how they address the problem of point distribution and difficulties with boundaries in problem (3.4).

Neighbor selection based on distance:

This is probably the easiest way to select neighbors for any point x_i in the domain. Taking into consideration the number of neighboring points needed to approximate the Laplace operator in a given dimension, neighbor selection for this case involves selecting n closest neighbors to a central point. In a 2d case, since there are 6 constraints given in (2.9), n needs to be 6 or greater to obtain a consistent stencil. The rest of the points can be assigned a weight of zero and thus not be included in the stencil calculation, speeding up the computational process as the resulting system matrix in (2.4) will be very sparse. However, depending on the spread of the points in the domain, this neighbor selection process can result in a lopsided distribution of neighboring points around a central point. Figure 3.7 demonstrates this when the 2d domain in problem (3.4) has central points lying very close to the boundary points.

Neighbor selection based on an inverse convex hull:

Neighbor selection in this case starts with taking the central point x_i as the origin and mirroring all other points at the unit circle so that points that lie inside the unit circle go outside and those on the outside go inside. We use the following relation for

the new points z_j that are mirrored from points x_j

$$z_j = x_i + \frac{x_j - x_i}{||x_j - x_i||^2}. \tag{3.9}$$

Through this inverse mapping, the original closest points to the central point have mirror points that are furthest away. A convex hull is then constructed to the mirror points. These mirror points that span the convex hull correspond to the original points that are taken as neighbors.

This method of neighbor selection is more expensive that just going by distance. However, there are fast convex hull algorithms in 2d and 3d [11, 23]. There is the advantage of guaranteeing that no point in the domain will have all of its neighbors lying on one side. On the other hand, depending on the original distribution of points, this method can result in too few neighbors selected than needed to approximate the Laplace operator. This is demonstrated in Figure 3.8.

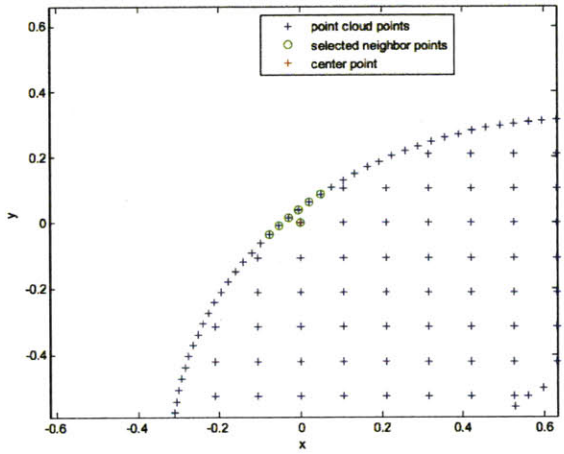


Figure 3-7: Selected neighbor points all lie on one side of the central point near the boundary

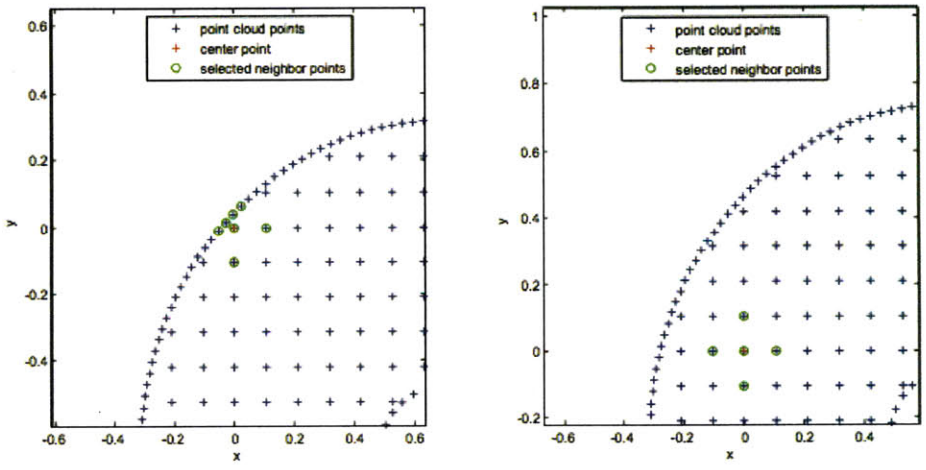


Figure 3-8: Selected neighbor points are distributed around the central point (a) but a central points can have too few neighbors selected (b)

3.2.1 Boundary Test

Using the two neighbor selection approaches from the previous section and considering the problem of central points near the boundaries having poor distribution of points around them as in Figure 3.7, we test out the inverse convex hull approach on all points next to the boundaries in 2d. The remaining interior points are set on a regular grid and we select the 6 closest neighbors for their finite difference stencil calculation. The total number of points in the point cloud for this test is set to 500 points. The results are shown in Figure 3.9 where plot (a) is generated using neighbor selection based strictly on distance and plot (b) is generated using the inverse convex hull approach on points near the boundaries. There are obvious irregularities seen in Figure 3.9(a), arising from the poor distribution of points near the boundaries; these irregularities can be seen at the $u = 0$ mark and at $u = 0.5$. In Figure 3.9(b), points that lie near the boundaries have 6 or more neighbors selected by the inverse convex hull approach. At most, 8 points span the inverse hull for these points.

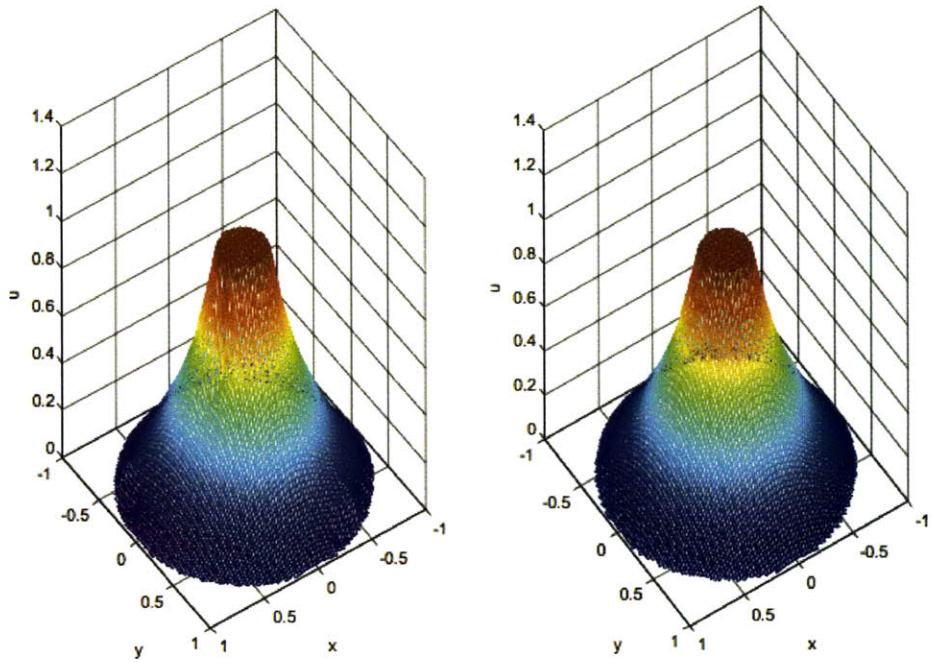
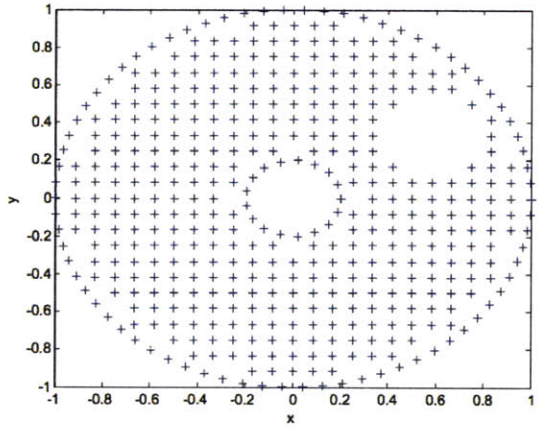


Figure 3-9: Solution of a 2d system using distance (a) and combination of distance and inverse convex hull (b) to set neighborhood criteria

3.2.2 Point Distribution Test

We also look at the need to include different neighborhood criteria when dealing with a gap inside the domain. If the 2d domain is discretized as shown in Figure 3.10(a), using distance to find the closest neighbors will result in poor neighbor selection. In this case, points near the gap are treated with the inverse convex hull for neighbor selection while only 6 closest neighbors are selected for stencil calculation of the remaining interior points. The results of the two systems are shown in Figure 3.10 (a), (b): one that uses only 6 closest neighbors and one that uses the inverse convex hull approach for points near the gap. Clearly using a neighborhood criteria based strictly on distance is insufficient for this case. A deformity can be seen in Figure 3.10 (a) that corresponds to the location of the hole in the domain. Poor point distribution leads to an inconsistent approximation.



(a)

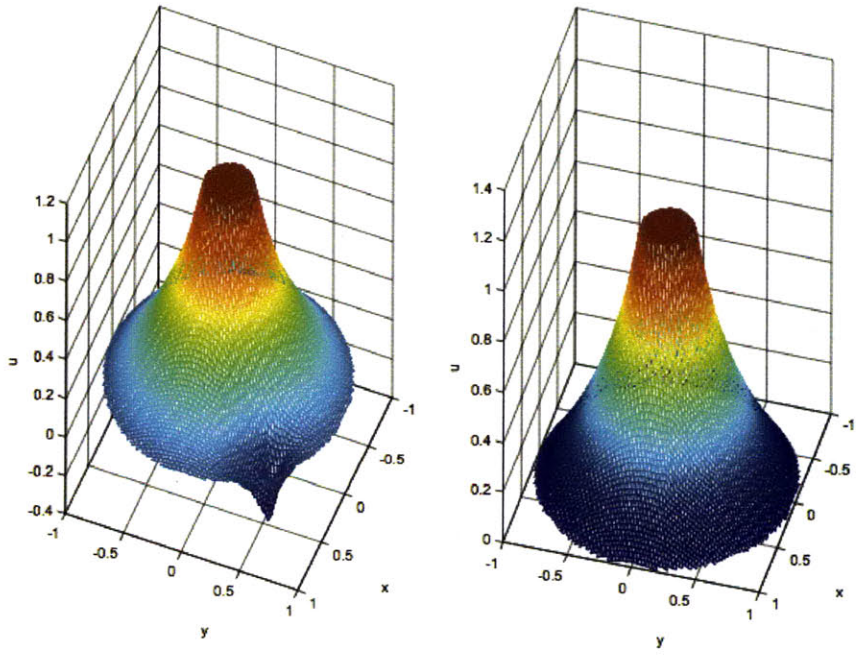


Figure 3-10: Computational domain with a gap in 2d (a) and the resulting solutions using distance (b) and inverse convex hull (c) to set neighborhood criteria

3.2.3 Robust Neighbor Selection in 3d

Large systems need an efficient way to handle neighbor search so as to make meshfree finite difference favorable over standard finite difference or other grid-dependent methods. For our 3d solver, we look to do a range search for the neighboring points around a central point with a range that always included more neighbors than necessary for a consistent approximation, which is at least 10 neighbors in 3d. This guarantees that every point will have enough neighbors selected. Furthermore, we select a range large enough so that there is some distribution of points around the central point. However, this means that we are always using a relatively large number of neighbors when dealing with a fine mesh. This requires a certain amount of data management. A naive nearest neighbor algorithm would scan an entire list of points in the domain, resulting in an $O(N^2)$ effort, where N is the size of the point set. To avoid this, we utilize a k-dimensional tree data structure to find the neighbors in a given range[1]. Our nearest neighbor algorithm operates on a kd-tree. The advantage of a range search on a kd-tree data structure is that it reduces the effort to be logarithmic in N for a fixed range size [18]. The complexity of the range search operation is generally $O(N \log N)$. We set the range size for our 3d solver to be 3 times the average mesh size. Furthermore, when the solver is used in a time-stepping scheme with little shift in the position of particles between time steps, the kd-tree structure from one time step can be used for the next time step, thus reducing the time it takes to build the tree.

3.3 Linear Solver Study

The resulting system matrix that results from the meshfree finite difference formulation in Chapter 2 is very sparse and non-symmetric. To solve the resulting large system in our 3d solver, we use preconditioned Krylov-subspace methods that are provided by

[1] A kd-tree is a binary tree used to store a finite set of points from a k-dimensional space. A pivot and a splitting plane are selected from which the root of the tree is built. In 3d, its construction involves splitting the points based on x-coordinate, then on y-coordinate, then on z-coordinate, and so on in alternating fashion. Studies of this algorithm can be found in [5, 12].

the PETSc library. We test various solvers provided by the library since depending on the problem at hand, there are always some solvers that outperform others. Furthermore, preconditioning is a key element when dealing with non-symmetric, and relatively ill-conditioned matrices; we therefore also test out the various preconditioners provided by the PETSc library. We selected iterative methods BiCGSTAB [25] and GMRES (general minimal residual method) [28] with various restart parameters and preconditioners block Jacobi and additive Schwarz, all provided by the software. Table 3.2 shows the number of iterations that various combinations of linear solvers and preconditioners need to converge to a solution for a 3d problem described in (3.2) for a system of 225,825 points. The stopping criterion for all cases is when the relative decrease in the residual norm is 10^{-4}. Table 3.2 also shows the performance of the iterative solvers with various preconditioners when the system is solved using a different number of processors. PETSc is set up to be executed in a parallel environment and uses MPI for its message-passing communication. Depending on the type of solver used, the parallel solver can be slower than the serial one. This mainly has to do with the communication time needed between the processors being greater than the amount of work each processor is actually given to do. This is further addressed in section 3.4. Most preconditioners provided by the software need more iterations before converging when used over more processors. This is particularly seen with the block Jacobi preconditioner. For multiple processors, the PETSc implementation of the block Jacobi preconditioning sets one block per processor and uses ILU (incomplete lower-upper) factorization to solve each block. Based on the number of iterations in Table 3.2, the best combination of iterative solver and preconditioner is the BiCGSTAB and additive Schwarz. This combination also scales best for more processors.

3.4 Parallel Efficiency

We further test our 3d solver for parallel efficiency. We use the MPI standard for all message passing between processors. There are two parts to parallelize in the solver:

Linear Solver/PC	itr (1proc)	itr (4 proc)	itr (16 proc)	itr (32 proc)
GMRES(30)/Block Jacobi	41	53	87	128
GMRES(30)/ASM	43	50	62	70
BiCGSTAB/Block Jacobi	31	37	69	89
BiCGSTAB/ASM	31	32	39	45

Table 3.2: Number of iterations for various combinations of iterative solvers and presconditioners

one is the main algorithm that determines the meshfree finite difference stencil as defined by (2.20) for each point in the domain, and the other is the setup of the stencils in a system matrix and solving the final system in (2.4). The first part is at best divided between the processors based on the total number of points for which finite difference stencils need to be determined. This generally results in all processors working on about the same number of points. The second part concerning the global system matrix is done automatically by the PETSc library. For p number of processors, parallel efficiency is computed by

$$E = \frac{T_{ser}}{pT}, \tag{3.10}$$

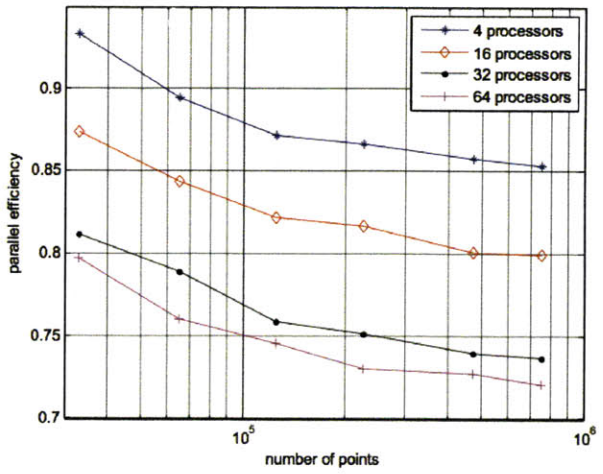
where T_{ser} is the runtime of the code in serial and T is the runtime of the code for p number of processors. Figure 3.11 shows the parallel efficiency for the stencil setup, matrix assembly and solving the linear system, and for the entire program.

The parallel efficiency of the stencil setup is high but not at 100 percent and slightly drops as more processors are added. Even though in general the number of points each processor is assigned is the same, the efficiency drops due to the number of neighbors given for each point in the domain. The range search described in section 3.2.3 does not guarantee that for every central point there will be an equal number of neighbors. In fact, for a random point distribution setting, most central points will have a slightly different number of neighbors. This is particularly true when comparing points near the boundaries and those further away. It is easy to see that points near the boundaries will generally have less neighbors given by the range

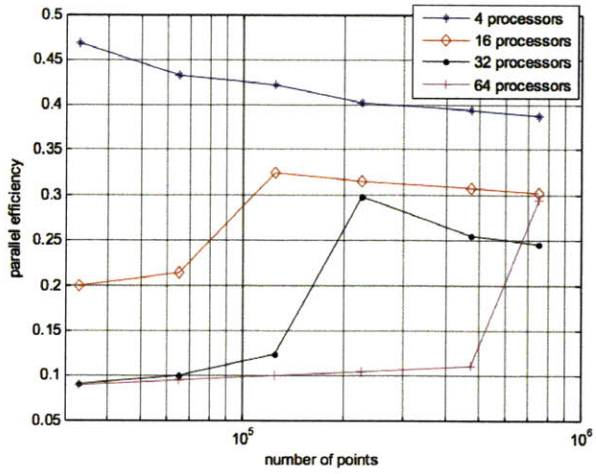
search. To optimize data management, the solver is set up in such a way that only the central point and its selected neighbors are carried through for stencil calculation, meaning that central points with more neighbors will have larger V and W matrices and thus more operations. As a result, even though every processor is in charge of the same number of points, some processors have to perform a greater number of calculations if the points they are given have a larger set of neighbors. The uneven amount of work each processor has to do only increases when more processors are added. This results in a drop in efficiency.

Unlike with the setting up the meshfree stencil, setting up the system matrix and solving it using the PETSc library does not result in a high parallel efficiency. Computing, communication, and memory requirements are different when using this software in serial and in parallel. Every processor needs to be in charge of a relatively large number of points (>20,000) for there to be a meaningful parallel speedup. This can be seen in Figure 3.11(b), where the parallel efficiency sharply increases when the problem size is increased to a certain point for a specific number of processors. Figure 3.11(c) shows the overall parallel efficiency of the solver with a sharp increase in the efficiency when using 64 processors on a larger data set. As seen in the PETSc parallel efficiency case for using 16 and 32 processors (Figure 3.11 (b)), after the sharp increase the efficiency slightly decreases and levels off. The same trend would most likely occur when using 64 processors. This means that using more processors will generally lower the overall parallel efficiency of the solver.

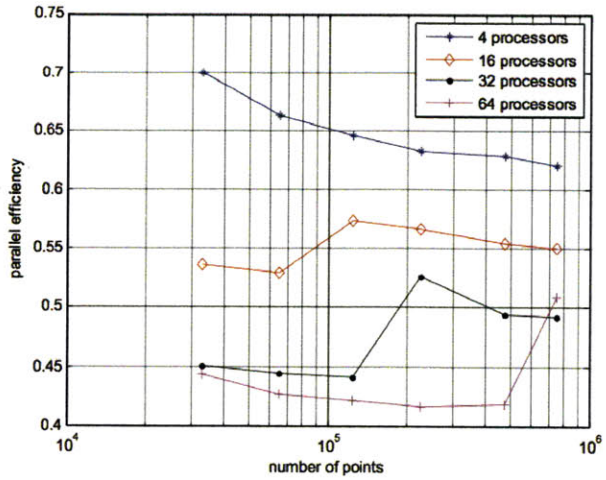
We looked to compare our documented performance of using the PETSc library to similar problems found in literature. Because most of the studies we looked at have implemented the PETSc linear solvers on either smaller problems than the ones we are testing with our solver or those that are symmetric, we focused on the trend of the parallel results rather than the actual parallel efficiency. The general trend is the same: for smaller problems, parallel efficiency decreases with an increase in the number of processors. For better efficiency, each processor needs to be assigned a relatively large scale problem [6, 14].



(a)



(b)



(c)

Figure 3-11: Parallel efficiency for stencil setup (a), matrix assembly and solve using PETSc (b) and the entire solver (c)

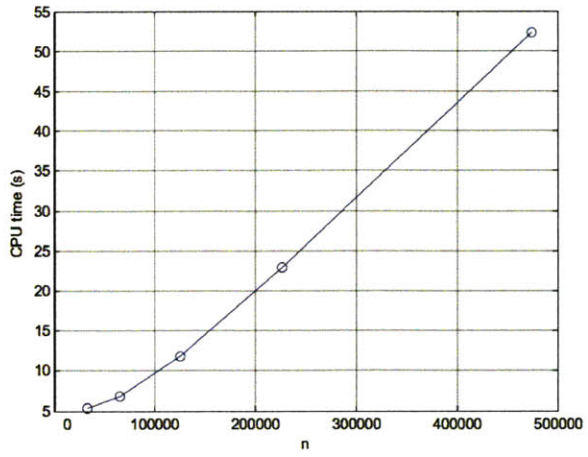
3.5 Computational Cost

Other than the need to develop neighborhood relations between points, the fundamental difficulty in mehsfree finite difference is overcoming the numerical effort that is required for the setup of stencils for differential operators. In any standard finite difference formulation, the stencils are known and can be hard coded. The computation effort of mehsfree finite difference stencil setup has been reviewed by Seibold [22]. For k number of constraints and m number of neighbors, the total number of flops is approximately $k(k+1)m + \frac{k^3}{3}$. The first term dominates for a well posed number of neighbors, resulting in a $90m$ effort in 3d. This estimation assumes that Cholesky decomposition is used to solve the $(V^TWV)^{-1}\mathbf{b}$ part of stencil equation (2.20). The matrix V^TWV is always symmetric positive definite and thus has numerous efficient options for solving systems of linear equations with it. Figure 3.12 shows the CPU time using one processor required for our 3d solver to setup the stencil and solve the arising system using PETSc. We use the BiCGSTAB method along with the additive Schwarz preconditioner to solve the system. Furthermore, profiling the PETSc performance shows that the majority of time is spent in applying the preconditioner and solving the system and not in setting up the PETSc system, even without using multiprocessors, meaning that time is not being spent in message passing between processors.

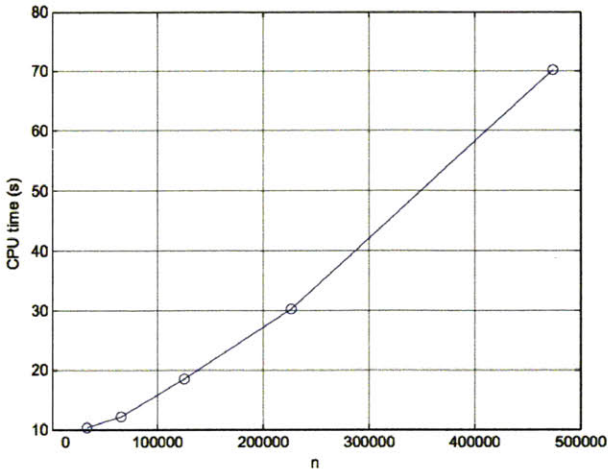
If a regular grid can be easily generated, then the computational effort for meshfree finite difference cannot compare to that of standard finite difference. Not only can the stencil matrix be hard-coded in standard finite difference, it is also symmetric. This is often not the case with meshfree finite difference. Convergence of the final system is thus less favorable with linear solvers in the meshfree case.

The advantage of meshfree finite difference mainly comes when dealing with complex geometry or discontinuity. In the case of problem posed in (3.2), a regular grid for standard finite difference cannot be imposed due to the boundaries. If encountered in a physical problem, options such as the Boundary Element Method (BEM) are available [3], but they might prove to be not very accurate, or expensive

and not very adaptive. Meshfree finite difference is a more adaptive method as shown in section 3.2 if the right neighbor selection approach is taken.



(a)



(b)

Figure 3-12: Computational cost of system setup (a) and solution of arising system (b) for 1 processor in 3d

Chapter 4

Application to Confined Flow

In this chapter we look at a wall-bounded flow problem. In particular, we simulate using a three-demensional vortex element method [7] the evolution of a vortex ring at an intermediate Reynolds number in a finite size box using our 3d meshfree Poisson solver to impose the normal flux boundary condition. In wall-bounded flows, the vorticity induced velocity field is computed as in an unbounded flow. The no-through flow boundary condition is accounted for by adjusting the velocity using the potential velocity generated by the wall-bounded potential flow. Various methods have been utilized to determine the potential velocity in confined flow problems, some grid based and some not [8, 9, 13]. In our case, by using a meshfree method for the potential flow, we keep the grid-free nature of the solution along with the Taylor expansion imposed accuracy of finite difference.

In what follows, we first formulate the numerical algorithm for the evolution of the vorticity field in a three-dimensional viscous flow. We then show the adaptivity of the meshfree solver in computing the potential velocity field to satisfy the normal velocity boundary conditions.

4.1 Numerical Method

We consider the three-dimensional incompressible Navier-Stokes equations in their vorticity-velocity formulation

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + \mathbf{u} \cdot \nabla \boldsymbol{\omega} - \boldsymbol{\omega} \cdot \nabla \mathbf{u} - \nu \Delta \boldsymbol{\omega} = 0, \tag{4.1}$$

where $\boldsymbol{\omega}$ is the vorticity field, $\mathbf{u}$ is the velocity field, ν is the kinematic viscosity, and ∇ is the gradient operator. The relationship between the vorticity and velocity is defined by $\boldsymbol{\omega} = \nabla \times \mathbf{u}$. The velocity field also satisfies $\nabla \cdot \mathbf{u} = 0$. Typically, a viscous splitting algorithm is used to solve the inviscid and the viscous parts of (4.1) in a time-stepping scheme [7]. That is, for every time step, we consider a convection step

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \boldsymbol{\omega} \cdot \nabla \mathbf{u} - \mathbf{u} \cdot \nabla \boldsymbol{\omega} \tag{4.2}$$

and a diffusion step

$$\frac{\partial \boldsymbol{\omega}}{\partial t} = \nu \Delta \boldsymbol{\omega}, \tag{4.3}$$

while retaining the advantages of Lagrangian vortex methods of avoiding the CFL requiements and suppressing numerical dissipation.

Convection Step:

We discretize the vorticity field using the vortex particle i with location χ_i, vorticity ω_i, and volume dV_i

$$\boldsymbol{\omega}(\mathbf{x}, t) = \sum_{i}^{N} (\boldsymbol{\omega}_i dV_i)(t) f_\sigma(\mathbf{x} - \boldsymbol{\chi}_i(t)), \tag{4.4}$$

where f_σ is a radially symmetric cutoff function having radius σ and is defined as $f_\sigma(\mathbf{x}) = \frac{1}{\sigma^3} f(\frac{|\mathbf{x}|}{\sigma})$. We take a low-order algebraic cutoff function as used in [20, 27]

$$f(r) = \frac{3}{4\pi} \frac{1}{(1 + r^2)^{5/2}}. \tag{4.5}$$

The convection step in (4.2) can be expressed in a Lagrangian formulation

$$\frac{d\boldsymbol{\chi}_i}{dt} = \mathbf{u}(\boldsymbol{\chi}_i). \tag{4.6}$$

This equation is solved using a second order predictor/corrector scheme. Evaluation of the velocity field is described in section 4.1.1.

Diffusion Step:

For the diffusion step in (4.3), we use the redistribution scheme proposed by Wee and Ghoniem [26], which is an extension of a vorticity redistribution method used in [24]. The key idea in this method is to redistribute the vorticity onto a grid using a modified interpolation kernel. The interpolation of a source particle at one time step is done to its nearest grid points lying on a uniform grid in a successive time step. That is, if we define f_{ij}^n as the fraction of the ith particle's strength that is transferred to the jth particle at the nth time step, the strength of the particles at time step $n+1$ is

$$(\boldsymbol{\omega}_j dV_j)^{n+1} = \sum_i^N f_{ij}^n (\boldsymbol{\omega}_i dV_i)^n. \tag{4.7}$$

The redistribution fraction is obtained through

$$f_{ij} = \Lambda_3(\frac{x_j - x_i}{\Delta x})\Lambda_3(\frac{y_j - y_i}{\Delta x})\Lambda_3(\frac{z_j - z_i}{\Delta x}), \tag{4.8}$$

where the interpolation kernel Λ_3 is defined as

$$\Lambda_3(\xi, c) = \begin{cases} 1 - 2c^2 + |\xi|(3c^2 - \frac{1}{2}) - \xi^2 + \frac{|\xi^3|}{2} & |\xi| < 1, \\ (2 - |\xi|(\frac{1}{6}(3 - |\xi|)(1 - |\xi|) + c^2) & 1 \leq |\xi| < 2, \\ 0 & 2 \leq |\xi|. \end{cases}$$

The variable c in the interpolation kernel is the diffusion length scale to grid size ratio, $c = \sqrt{\nu \Delta t}/\Delta x$. This completes the diffusion step, reducing the remeshing and diffusion processes into one step.

As the velocity normal to each wall is zero means that particles should not be able

to cross the boundaries. During the diffusion step, however, the algorithm permits some particles to cross over. To counteract this, we take the closest particles that diffused outside the boundaries and add their respective vorticity components to the vorticity components of their mirror images lying within the boundries. We use all particles that lie $3\Delta x$ outside the boundaries, where Δx is the diffusion step grid size.

4.1.1 Velocity Evaluation

We use the Helmholtz decomposition to describe the velocity field. This description includes the rotational component $\mathbf{u}_w$ of the velocity field that accounts for the vorticity in the flow and the potential component $\mathbf{u}_\phi$ that is used to enforce the boundary conditions. The presence of potential velocity also ensures that the boundaries are compatible with the velocity field along with the vorticity field. The Helmholtz decomposition is defined by

$$\mathbf{u} = \mathbf{u}_\omega + \mathbf{u}_\phi. \tag{4.9}$$

The potential velocity component can be determined from the potential function ϕ by the relation $\mathbf{u}_\phi = \nabla\phi$. The potential is a scalar field whose relationship to the velocity vector satisfies irrotationality. The velocity can also be expressed in terms of a stream function $\boldsymbol{\psi}$, whose relationship to the velocity vector satisfies continuity. The velocity field can be rewritten as

$$\mathbf{u} = \nabla \times \boldsymbol{\psi} + \nabla\phi. \tag{4.10}$$

Furthermore, if we define Ω and Γ as the fluid domain and the boundary surface, respectively, than the relationship between the stream function $\boldsymbol{\psi}$ and vorticity $\boldsymbol{\omega}$ is

$$\begin{aligned} -\Delta\boldsymbol{\psi} &= \boldsymbol{\omega} \quad \text{in } \Omega \\ \Delta \cdot \boldsymbol{\psi} &= 0 \quad \text{in } \Omega. \end{aligned} \tag{4.11}$$

The potential ϕ needs to satisfy

$$\begin{aligned} \Delta\phi &= 0 \quad \text{in } \Omega \\ \frac{\partial\phi}{\partial\mathbf{n}} &= -(\nabla\times\boldsymbol{\psi})\cdot\mathbf{n} \quad \text{on } \Gamma. \end{aligned} \tag{4.12}$$

To evaluate the vortical velocity $\mathbf{u}_\omega$ along with velocity gradient $\nabla\mathbf{u}_\omega$ that are needed in the convection step, we use a multi-purpose adaptive tree-code proposed in [20]. The vortical velocity is determined by the Biot-Savart law

$$\mathbf{u}_\omega(\mathbf{x},t) = -\frac{1}{4\pi}\int_\Omega \frac{(\mathbf{x}-\mathbf{x}')\times\boldsymbol{\omega}(\mathbf{x}',t)}{|\mathbf{x}-\mathbf{x}'|^3}d\mathbf{x}'. \tag{4.13}$$

The tree-code uses the Rosenhead-Moore kernel to rewrite the velocity as

$$\mathbf{u}_\omega(\mathbf{x},t) \approx \sum_i^N -\frac{1}{4\pi}\frac{\mathbf{x}-\boldsymbol{\chi}_i}{(|\mathbf{x}-\boldsymbol{\chi}_i|^2+\sigma^2)^{3/2}}\times\boldsymbol{\omega}_i dV_i. \tag{4.14}$$

The computational effort of the adaptive tree-code is about $O(N\log N)$, where N is the total number of computational elements. The potential velocity $\mathbf{u}_\phi$ and $\nabla\mathbf{u}_\phi$ are computed using the meshfree finite difference formulation in Chapter 2 for the equations in (4.12). The two velocity fields are added to represent the total velocity field $\mathbf{u}$ in (4.2).

4.2 Numerical Example

We now look at simulating a single vortex ring in a wall-bounded domain at an intermediate Reynolds number $Re = 500$. This is an extension of the vortex ring simulation done in [20, 26]. The vortex ring is of radius R and core radius a. The vorticity of the core of the ring is set to

$$\omega_\phi = 10\frac{K}{\pi}\frac{\Gamma}{a^2}\exp[-K(\frac{R^2}{a^2}+\frac{r^2}{a^2}-\frac{2Rr}{a^2}\sin\theta)]. \tag{4.15}$$

Here, $r = \sqrt{x^2+y^2+z^2}$, $\tan\theta = \frac{\sqrt{x^2+y^2}}{y}$, $K = \frac{(2.24182^2)}{4}$, $\frac{a}{R} = 0.35$, circulation $\Gamma = 1$ and $R = 1$. Following the interpolation in the diffusion step, the problem size is controlled by removing particles whose strength is less than a specified limit, $|\omega_i dV_i| < 10^{-11}$.

We first form the boundaries of a finite size box and place the vortex ring in its center. We use a cube with a side length of 6 centered at the origin. The grid size for the boundaries and the diffusion step is set to $\Delta x = 0.1$. The boundaries of the cube remain the same throughout the simulation. The grid generated by the diffusion step is used in the convection step to solve for the potential function in (4.12) using the meshfree Poisson solver with Neumann boundary conditions. With each time step, we retain the solution of the potential function from the previous time step as the starting value of the iterative procedure that solves the system in (2.4). Figure 4.1 shows isosurfaces of vorticity magnitude at various time steps, capturing the collision of the ring with the wall.

To validate the potential velocity and its gradient results obtained using the meshfree solver, we use the image method to treat the effects of the 6 walls of the cube during each convection step. For the wall positioned at z=3, for instance, the image method results in placing an image vorticity at

$$x_{img} = x, \quad y_{img} = y, \quad z_{img} = 3 + (3 - z), \tag{4.16}$$

with the magnitude of vorticity of

$$\omega_{img,x} = -w_x, \quad \omega_{img,y} = -w_y, \quad \omega_{img,z} = w_z. \tag{4.17}$$

This procedure is repeated for each wall. Furthermore, to account for the edges and corners of the cube, extra images are placed across each edge and each corner. In total, 26 images are used to guarantee that the normal-velocity boundary conditions are enforced.

In Figure 4.2, the center of vorticity in the y direction is plotted for the two

cases. The maximum value is reached at the time of the collision of the vortex ring with the wall. Figure 4.3 shows the speed of the vortex ring in the y-direction. To better visually compare the two cases, Figure 4.3 also shows the vorticity contours for various time steps. The two methods show a difference in the location of the center of vorticity and speed after the collision of the vortex ring with the wall. However, both methods also show to have similar errors in correctly calculating a zero normal velocity on each of the boundaries after the collision of the vortex ring with the wall, with the image methods faring slightly better. Table 4.1 shows the breakdown of the maximum absolute error between the normal velocity on the $y = 3$ face of the cube as calculated using the two methods and the expected zero normal velocity.

time	max. abs. error: image method	max. abs. error: meshfree method
t = 5.85	5.25 E-02	5.68 E-02
t = 7.35	3.69 E-02	3.97 E-02
t = 10.35	2.22 E -02	2.41 E-02

Table 4.1: Absolute error of normal velocity condition on the $y = 3$ face of the cube after collision of vortex ring for the image method and meshfree method

The image method for this case lacks accuracy as more images might be needed in the calculation of the potential velocity. With the meshfree method, the accuracy is lost in the calculation of the potential function. We use a rather coarse grid in this example with a tolerance of 10^{-4} in the relative decrease in the residual norm as a stopping criteria for iterative solver for the global system matrix. A smaller tolerance would mean a more accurate potential function. However, in the tested cases, lowering the tolerance causes the BiCGSTAB method to fail to converge in 10,000 iterations for various time steps. The accuracy of the meshfree solver does improve when for a more refined grid. Table 4.2 shows the maximum absolute error on the $y = 3$ face of the cube after one time step for various mesh sizes.

mesh size	maximum absolute error
0.12	9.83E-05
0.1	7.43E-05
0.08	3.03E-05
0.05	1.93E-05

Table 4.2: Absolute error of normal velocity boundary condition on the $y = 3$ face of the cube after one time step for the meshfree method

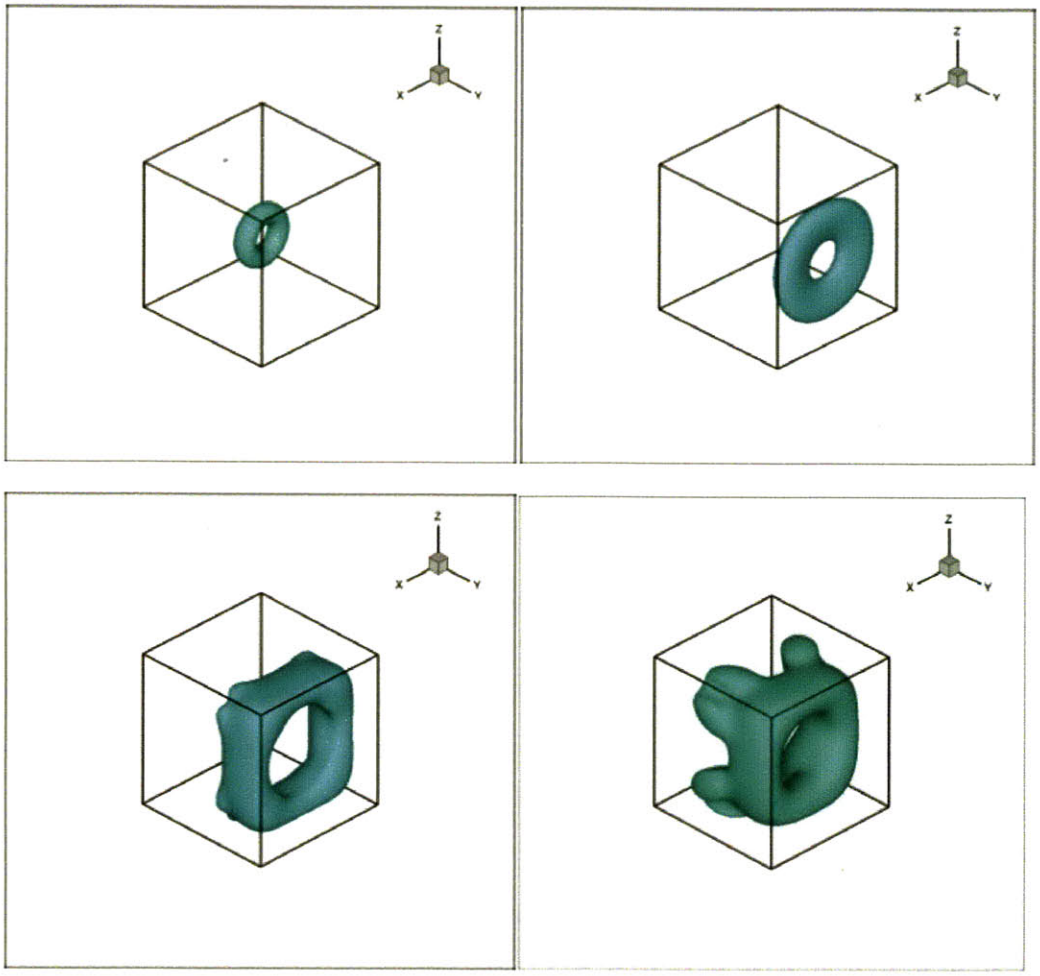


Figure 4-1: Isosurfaces of vorticity for times t = 0, 2.85, 7.35, and 13.35

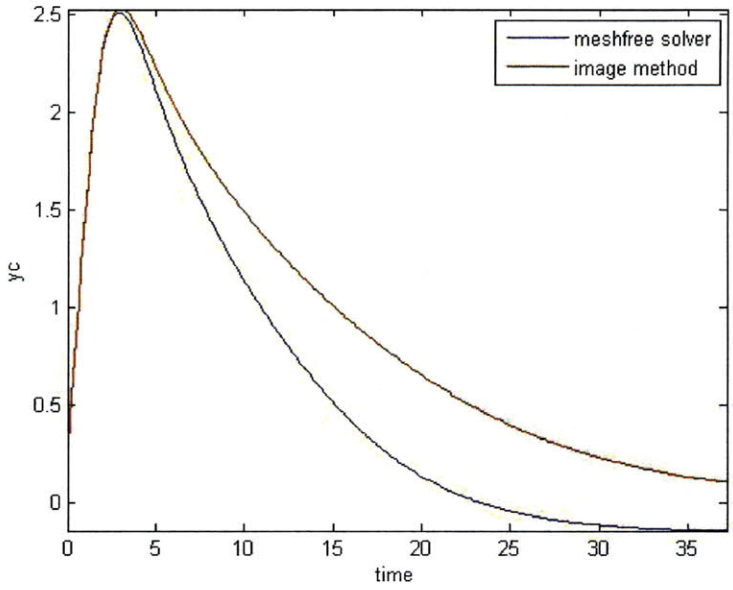


Figure 4-2: Time history of center of vorticity along the y-axis

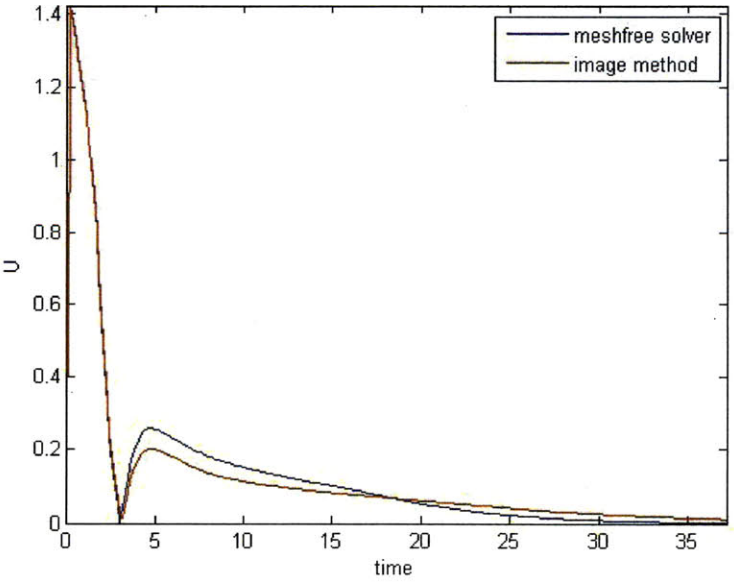
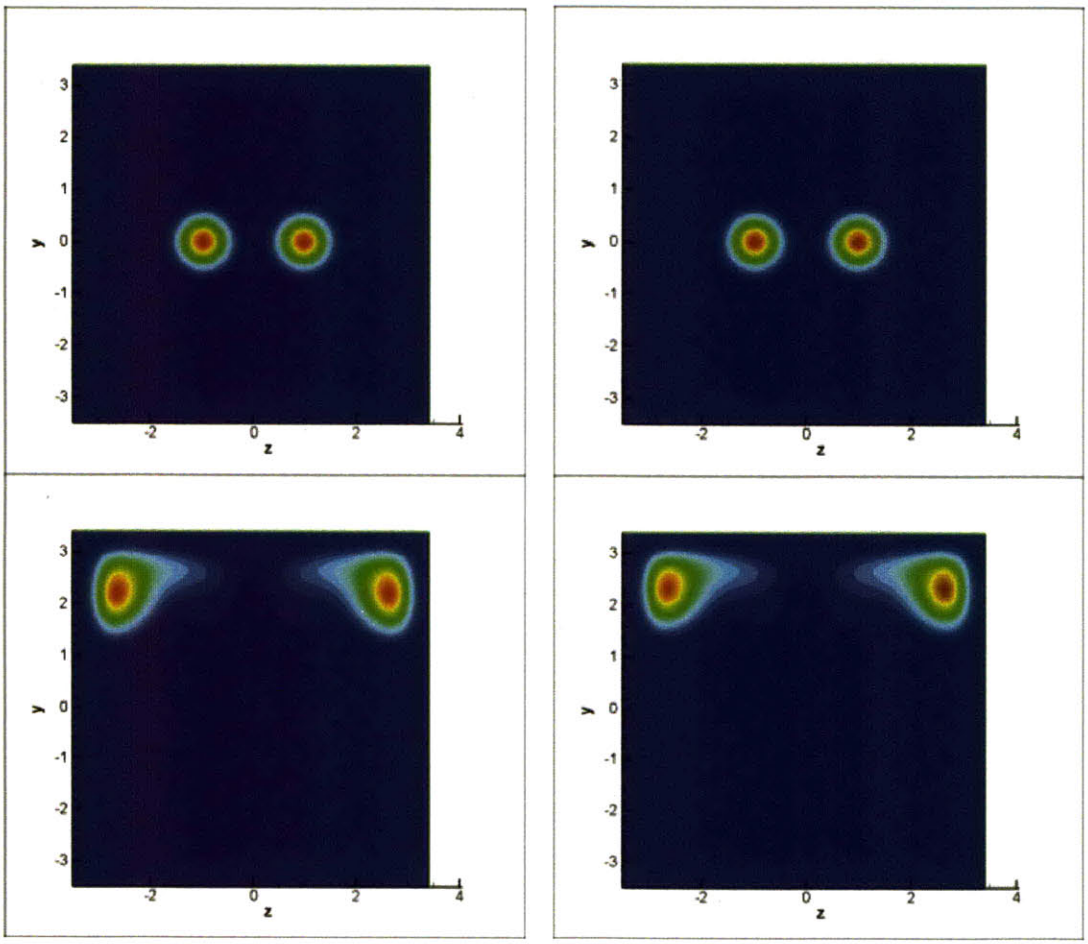


Figure 4-3: Time history of velocity along the y-axis


y
z
3
2
1
0
-1
-2
-3
-2
0
2
4

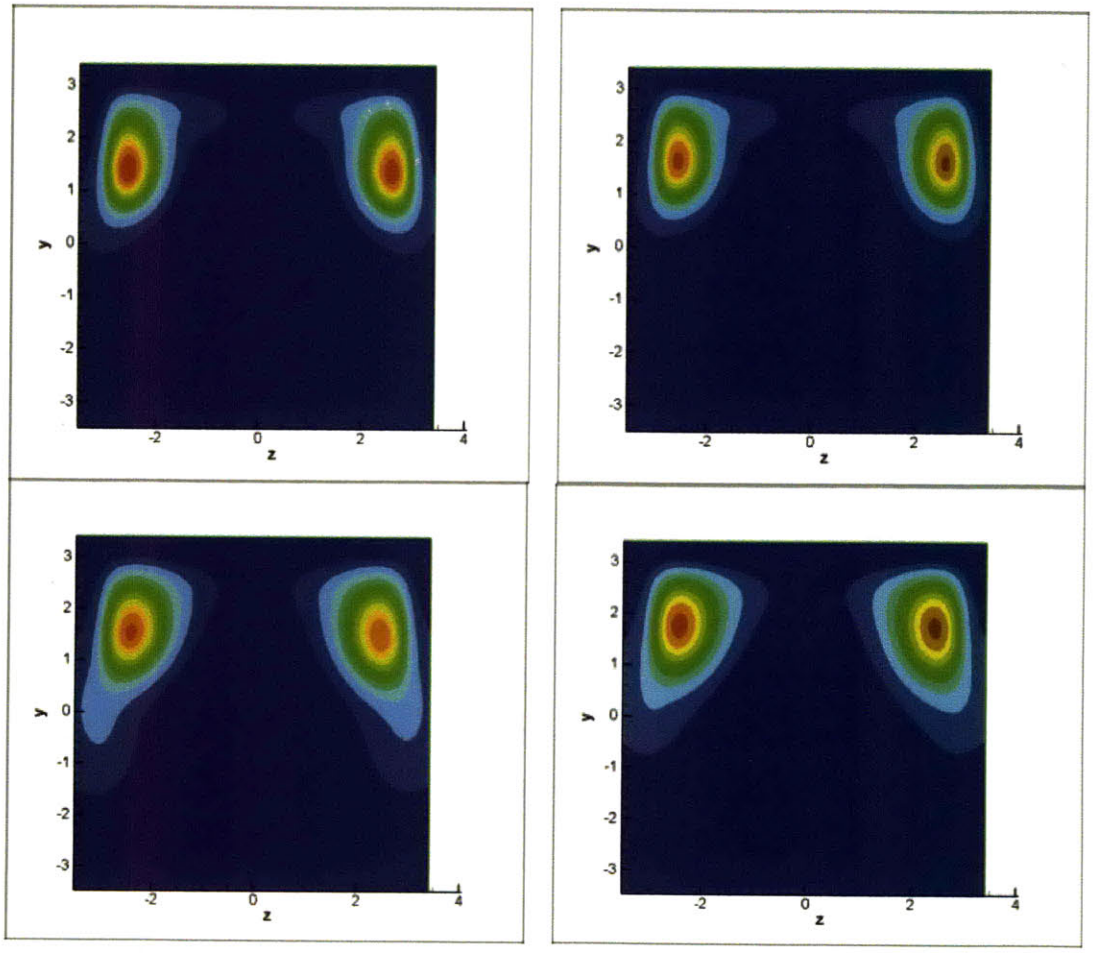


Figure 4-4: Vorticity contours for using meshfree solver (left) and the image method (right) for t = 0, 4.35, 8.85 and 13.35

Chapter 5

Conclusions

A meshfree finite difference solver has been described, tested, and implemented in a wall-bounded flow simulation to treat boundary conditions. The solver is based purely on nodal points without the need to prespecify node connectivity, as is required with traditional meshing. The point distribution for the meshfree finite difference approach does not need to be regular. The method implements a local iterative weighted least squares approximation using neighbor relations to compute finite difference stencils. The resulting stencils are not guaranteed to be positive and form nonsymmetric system matrices for the cases where regular point distribution are not desired. For large problems with complex geometries, these matrices are often ill-conditioned. Various options for dealing with the global stencil matrices have been shown through the use of the parallel PETSc library.

The convergence of the method is showed to depend on the selected weight function as well as the average mesh size. The accuracy and numerical convergence of the solver are shown using a circular domain with both irregualr and regular distribution of points. Further examples in 2d are used to examine the solver's adaptivity when dealing with domain discontinuities and inconsistent distribution of points near the boundaries. It has been noted that the method requires a certain degree of adequite point distribution and enough neighbors to guarantee a consisten approximation. Keeping in mind that the method can get quite expensive without proper data management, this can only be achieved by having an idea of how the points are

distributed within the domain before using the solver.

Meshfree methods have been studied for incompressible flow problem applications, particularly in simulations by the Lagrangian particle method where particles can be used as gird points for the meshfree method. This makes them well suited for vortex methods. The 3d mehsfree solver developed in this work has been applied to an intermediate Reynolds number confined flow simulation as an add on to the vortex element method. The meshfree solver is used to treat the effects of the boundaries and satisfy the normal flux boundary condition. This requires an accurate evaluation of the potential function used to calculate the potential velocity and its gradients, posing a Neumann boundary problem. The results from using the meshfree solver to treat the potential velocity has been compared to those produced using the image method. The two methods show differences in the location of the center of vorticity and speed after the collision of the votrex ring with the wall. In terms of the meshfree method, the accuracy of solver might have been compromised by the low grid resolution and the selected stopping criteria for the iterative method used to solve the global system for the potential function. Further work on the solver would include running the confined flow simulation in parallel so that a finer mesh can be used and its effect on the numerical convergence can be studied.

Finally, at the current stage it is difficult to make any real assumptions as to how this meshfree solver will fare against other methods, for instance Panel methods [17], aimed at calculating the velocity potential in wall-bounded flows. Choosing the right approach greatly depends on the problem and geometry at hand. The meshfree finite difference method is adaptive however, and can be applied to any step where using a mesh is troublesome, a relatively high order accuracy can be achieved, and data management can be done efficiently.

Bibliography

[1] S Balay, K Buschelman, V Eijkhout, W Gropp, D Kaushik, M Knepley, LC McInnes, B Smith, and H Zhang. Portable, extensible toolkit for scientific computation - version 3.0, 2009.

[2] S Balay, K Buschelman, V Eijkhout, W Gropp, D Kaushik, M Knepley, LC McInnes, B Smith, and H Zhang. Petsc user manual - revision 3.1. Technical Report ANL-95/11, Argonne National Laboratory, March 2010.

[3] P.K. Banerjee and R. Butterfield. *Boundary Element Methods in Engineering Science.* McGraw Hill, New York, 1981.

[4] T. Belytschko, Y. Krongauz, D. Organ, and M. Fleming. Meshless methods: An overview and recent developments. In *Computer Methods in Applied Mechanics and Engineering*, pages 3–47, 1996.

[5] Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[6] O. Ceylan and . Kalenderli. Parallel computation of two dimensional electric field distribution using petsc. *Lecture Series on Computer and Computational Sciences*, 1:1–3, 2004.

[7] G.-H Cottet and P.D. Koumoutsakos. *Vortex Methods: Theory and Practice.* Cambridge University Press, 2000.

[8] G.-H. Cottet and P. Poncet. Particle methods for direct numerical simulations of three-dimensional wakes*. *Journal of Turbulence*, 3:38–+, October 2002.

[9] G.-H. Cottet and P. Poncet. Advances in direct numerical simulations of 3d wall-bounded flows by vortex-in-cell methods. *J. Comput. Phys.*, 193(1):136–158, 2004.

[10] G.A. Dilts. Moving least squares particle hydrodynamics i, consistency and stability. *International Journal for Numerical Methods in Engineering*, 44(8):1115–1155, 1999.

[11] William F. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3(4):398–403, 1977.

[12] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[13] A. Gharakhani and A.F. Ghoniem. Bem solution of the 3d internal neumann problem and a regularized formulation for the potential velocity gradients. *International Journal for Numerical Methods in Fluids*, 24:81–100, 1997.

[14] P.A. Kler, E.J. Lopez, L.D. Dalcin, F.A. Guarnieri, and M.A. Storti. High performance simulations of electrokinetic flow and transport in microfluidic chips. *Comput. Methods Appl. Mech. Engrg.*, 198:2360–2367, 2009.

[15] J. Kuhnert and S. Tiwari. Finite pointset method based on the projection method for simulations of the incompressible navier-stokes equations. In M. Griebel and M.A. Schweitzer, editors, *Meshfree methods for Partial Differential Equations*, volume 26, pages 373–388. Springer, 2002.

[16] T. J. Liszka, C. A. M. Duarte, and W. W. Tworzydlo. hp-meshless cloud method. *Comp. Meth. Appl. Mech. Eng*, 139:263–288, 1996.

[17] J.L. Panel. Panel methods in computational fluid dynamics. *Annu. Rev. Fluid Mech. 1990*, 22:255–274, 1990.

[18] F.P. Preparata and M. Shamos. *Computational geometry: an introduction.* Springer Verlag, New York, 1985.

[19] A.G. Petschek P.W. Randles, L.D. Libersky. On neighbors, derivatives, and viscosity in particle codes. Technical Report LA-UR-99-2560, Los Alamos National Laboratory, Los Alamos, NM, September 1999.

[20] Fabrice Schlegel, Daehyun Wee, and Ahmed F. Ghoniem. A fast 3d particle method for the simulation of buoyant flow. *J. Comput. Phys.*, 227(21):9063–9090, 2008.

[21] B. Seibold. Minimal positive stencils in meshfree finite difference methods for the Poisson equation. *ArXiv e-prints*, February 2008.

[22] Benjamin Seibold. *M-Matrices in Meshless Finite Difference Methods.* PhD thesis, University of Kaiserslautern, Germany, 2006.

[23] Raimund Seidel. A convex hull algorithm optimal for point sets in even dimensions. Technical report, Vancouver, BC, Canada, 1981.

[24] S. Shankar and L. van Dommelen. A new diffusion procedure for vortex methods. *J. Comput. Phys.*, 127(1):88–109, 1996.

[25] H. A. van der Vorst. Bi-cgstab: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[26] Daehyun Wee and Ahmed F. Ghoniem. Modified interpolation kernels for treating diffusion and remeshing in vortex methods. *J. Comput. Phys.*, 213(1):239–263, 2006.

[27] G. S. Winckelmans and A. Leonard. Contributions to vortex particle methods for the computation of three-dimensional incompressible unsteady flows. *J. Comput. Phys.*, 109(2):247–273, 1993.

[28] Mi young Kim, Sin jae Kang, and Jong hyeok Lee. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific Statistical Computing*, 7:856–869, 1986.